$4.00

programmer's journal

The Resource Journal for IBM PC Programmers

Volume 3
Number 1

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

POWER TOOLS

from *Phoenix*

You are a serious programmer. Your income depends on developing software solutions for complex problems. Fast. Error free. You need powerful tools for your compiler. That's what we sell.

PLINK ™ 86
OVERLAY LINKAGE EDITOR

- overlays for modules are automatically loaded during execution
- generates symbol tables for PFIX 86 PLUS
- accepts unlimited number of files
- capacity for over 35,000 symbol, commonblock and variable names
- change overlay structure without recompiling
- sorted memory map reports
- user defined width/height for reports
- define symbols as absolute addresses or offsets from other symbols
- over loader is user configurable

List: $395

PFIX ™ 86 PLUS
DYNAMIC SYMBOLIC DEBUGGER

- debugs PLINK 86 overlays
- view source code while debugging
- set temporary or permanent breakpoints
- advanced tracing and trapping capability
- in-line assembler for temporary patches
- preserves program screen and supports dual CRT configurations
- multiple windows to view code, data, breakpoints, register/stack contents, simultaneously
- supports 8087 numeric coprocessor

List: $395

PMATE ™
TEXT EDITOR PROCESSOR

- automatic disk buffering
- 10 auxiliary buffers that can be edited
- garbage stack
- user customization
- 120 comand macro language
- store macros permanently
- arithmetic and logical operations
- 100 numeric variables
- automatic formatting with indent and word wrap
- insert, overwrite and command modes
- horizontal/vertical scrolling

List: $225

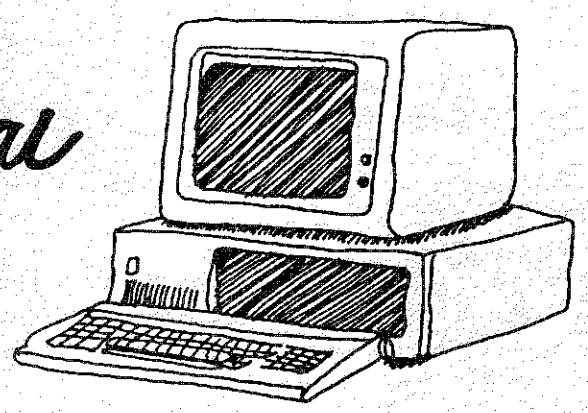
[PC]² We Sell Power

Phoenix Computer Products Corporation
1416 Providence Hwy., Suite 220
Norwood, MA 02062
800-344-7200 (Mass.) 617-762-5030

Visa • Mastercard • American Express

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal



CONTENTS

HOUSEKEEPING 2

LETTERS 3

PJ INTERVIEW: NOVEMBER'S COMDEX 4
Ronald Moak's view of the most recent Comdex

PJ FORUM: C — ITS USES AND ABUSES by Rex Jaeschke 7
A discussion of C as a commercial applications programming language

C: USING THE C PREPROCESSOR by Frank D. Greco, AGS Computers, Inc. 12
Speeding up your C functions

ASM: MEMORY SEGMENTATION EXAMPLES by Bill Salkin 14
Assembly code that illustrates Segment usage

BASIC: INDEX FILES (PART I) by Ronald Moak 20
Designing optimal file structures

TOOLS: STRUCTURED PROGRAMMING LANGUAGE by Dennis Baer 22
A step beyond Microsoft BASIC

BASIC: A COMMERCIAL PROGRAMMER LOOKS AT BASIC ON THE IBM PC by Bette Laswell 24
The founder of BLD Homeware evaluates Basic

DTACK: SURPRISING MEMORIES by FNE 28
Excerpts from DTACK Grounded

REVIEW: PERISCOPE, A SYMBOLIC DEBUGGER by Bill Salkin 34
New solutions to your debugging problems

BOOKS: FOUR PROGRAMMING TEXTS reviewed by J. Edward Volkstorf, Jr. 39
Which ones are worth the price

MISC. 43

ADVERTISERS' INDEX 43

ISSN#0747-5861

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOUSEKEEPING

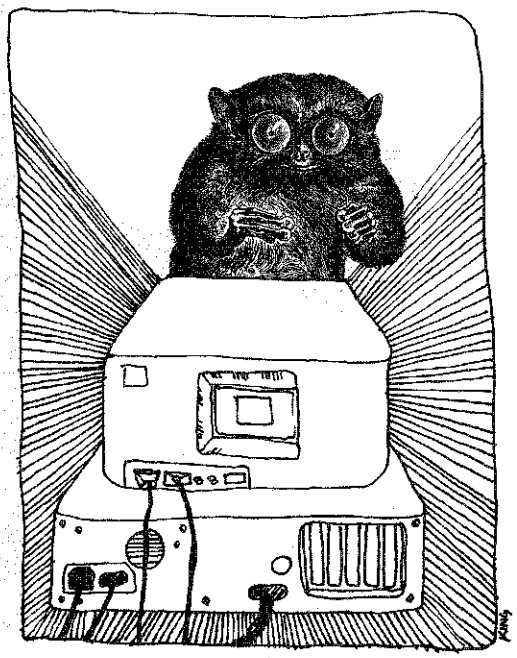
We have decided to drop our "read protection" scheme for PJ, called READPROLOCK. The special glasses you had to wear while reading PJ were too expensive to produce. We tried a sample run, delivered them to southern Oregon, and got a few complaints from subscribers. You see, if you attempt to read a pirated copy of PJ without the glasses, you suffer severe retina burn, which lasts three to six hours. This serves any pirates right for "borrowing" a copy of PJ that they haven't paid for. The legal hassles, from retina worms, loomed too large on the horizon, so we have finally dropped this particular scheme.

Don't think we've given up on this protection business. No way! There's big money to be made selling it to other magazine publishers.

Notes: We have heard quite a few compliments from programmers about Sidekick. The new IBM Enhanced Graphics board is great. 2400 baud modems are right around the corner (didn't we just get our BBS's up to 1200 baud?). Why not go directly to 9600 baud, do not pass GO, and skip the in between speeds? Yeah, I know it is a little difficult. My mother-in-law reached 9600 baud on standard phone lines years ago.

It would really be nice to have a Virtual Checking Account. Whenever any creditors needed money they could just swap in the amount they needed temporarily and then put it back in your account. You wouldn't actually have to pay your bills in full, and you could charge them an override fee for the money they ran in foreground from your account.

—Greg □



Video Operators Display Syndrome: Final Stages

programmer's journal

Publisher:	Elizabeth Oakley
Editor:	Greg Estes
Managing Editor:	Louise Shimmel
Copy Editor:	Marilyn Reeves
Production Manager:	Christine Baldwin
Production Assistant:	Jennifer Mullock
Cover Design:	Marilyn Reeves
Illustrations:	Julia King
Advertising Director:	Louise Shimmel
Contributing Writers:	Ron Moak Frank Greco Bette Laswell Dennis Baer Rex Jaeschke Bill Salkin FNE J. Edward Volkstorf Jr.
Published by:	Avant-Garde Publishing Corp.
Business Office:	Avant-Garde Publishing Corp. 1907 Garden Ave. Eugene, OR 97403 (503) 345-3043
Editorial Office:	2765 Potter St. Eugene, OR 97405 (503) 687-0398
Subscription Inquiries:	Programmer's Journal P.O. Box 3000 Dept. EE Denville, NJ 07834 (503) 345-3043
Postmaster send address changes to:	Programmer's Journal P.O. Box 3000 Dept. EE Denville, NJ 07834

Subscriptions: Bi-monthly rates are $24 U.S. ($30 in Canada, and $40 elsewhere)
IBM and Personal Computer are trademarks of International Business Machines, Armonk, New York; dBASE II is a trademark of Ashton-Tate.
Published bi-monthly.

Copyright ©1984
by Avant-Garde
Publishing Corp.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

PUSHING AND STACKING

Congratulations! Volume 2, #5, was your best yet! Keep up the good work.

I just wanted to comment on the article by Karl F. Thompson called "BASIC: Scroll Control Using BASIC and Assembler. Adding Finesse to Your Programs."

He correctly states (p.18, column 2 mid-way down), "(One note of caution: the Stack that is created by BASIC's CALL, has for its default room to store only 16 bytes.)"

This means that all you have room for is space enough for: 8 PUSHES, or 7 PUSHES and 1 NEAR CALL, or 6 PUSHES and 1 FAR CALL, or 5 PUSHES and 1 INT, etc.

Thus we see two problems in his assembler sub-routine listing, pages 23-25. He uses 5 PUSHES, 1 NEAR CALL and 1 INT. This overflows the allotted Stack.

But what is most often overlooked (and I've seen this happen a number of times before in both your journal and some others) is that the BIOS (called INT 11H and 10H through this program) uses YOUR Stack. It does not establish one of its own. A quick inspection of the code for INT 10H shows no fewer than 8 PUSHES when it starts up.

But that's hardly the end of it!! Since the first BIOS instruction for INT 10H is an STI, this routine can be interrupted by another BIOS routine such as KB_INT, or TIMER_INT, or RS-232 interrupt-driven routines.

Are you running a Spooler? There's another one.

So guess what? You may allocate 50 bytes of Stack space and think that's perfectly safe and still get burned!

Since BASIC's Stack is at the top of its Data Segment, and since an overflow downward would take it into its string area, and since this area is not usually used except by very large programs or ones that do a lot of string maintenance, you're unlikely to experience problems, except maybe once in a long, long while.

How many "unexplained glitches" which "occur sporadically" are caused by an insufficient Stack space? Who knows, but I'll bet it's a lot, having seen as many assembler listings as I have. "It works great except when I try to spool my printout," etc. etc.

Moral: Consider a 64-byte Stack absolutely bare bones and in fact fairly risky. I use a 256-byte Stack always and have never regretted it. Remember, as more interrupt-driven resident routines proliferate, your small Stack gets closer and closer to the edge.

So my advice is to use a 256-byte Stack and be done with it. Heck, in 3 years you may need a 512-byte Stack to work with state-of-the-art software.

In the assembler sub-routines shown in the article, it is certainly best to allocate its own Stack early in the processing and restore BASIC's Stack pointer just before returning.

Chris Rivers
Software Synthesis
Oakland, CA

NEW OP-CODE

In 1975, several systems programmers and I made up OP-CODES to add to the already rich set available on the DECsystem-10. The only one I remember worth mentioning was:

BGTL - Branch on operator Gone To Lunch

Andrew R. Farber
Tampa, FL □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NOVEMBER'S COMDEX

Ron Moak, Programmer's Journal columnist, takes a look at Comdex

PJ: How many computer faires have you attended?

RON: Several IBM PC conventions, including the West Coast Faire and the PC Faire in San Francisco.

PJ: From just an IBM PC viewpoint, is going to Comdex worth the hassle and expense?

RON: Comdex is basically a dealer, OEM, and VAR (Value Added Reseller) convention, even though end users are definitely welcome. There is such a large number of booths it is almost impossible to see them all in the time allotted. If you are strictly interested in IBM PC gear and have no desire to see anything else, you might do better attending one of the PC computer faires. However, if you have any inclination to see newly announced products or find out how the PC is being used in micro/mainframe connections, then a trip to Comdex is very worthwhile. Anyone curious about new hardware, the PC-compatible market, and multi-user systems should attend, too. You get a good look at where the whole computer business is moving.

PJ: Which operating systems seemed prevalent?

RON: MS-DOS was there as well as PICK and a number of others. Unix appeared to be the dominating force as far as I could tell. There were a lot of applications running under Unix or Xenix on a number of different machines. For single users, most of the booths were using MS-DOS or Macintosh systems. I saw a lot of Macintosh software. Multi-user systems were predominately Unix or Xenix. There was a PICK system running on a regular IBM PC with two terminals hooked up to it.

PJ: Does REVELATION run under MS-DOS or PICK?

RON: REVELATION runs under PICK. The PICK operating system is designed around database management and information handling, so writing an information manager with PICK is a natural and straightforward exercise.

PJ: Did you see TopView?

RON: Yes, I spent some time looking at it. The person demonstrating it at Comdex was one of the programmers who helped write it. He had a lot to say about it. He discussed the time-splicing features when several jobs are running. The keyboard is always attached to the active, foreground job. There is no delay when you type in commands from the keyboard.

The demonstration was running three jobs on an XT rather than an AT. This was more impressive, as we naturally expect things to really bog down on just the PC/XT. You could see all three partitions in separate windows as they were running. There seemed to be no way to set partition priority or allocate processing power. The programmer who was showing TopView had a database manager in one window, a numeric counter in another window, as well as the foreground task.

PJ: Would TopView work on a floppy-based IBM PC?

RON: TopView takes up a lot of room. They say it will run on a floppy-based machine, but a hard disk is more feasible. On a 256K system, 80K is left for applications. There is a lot of overhead. TopView takes all the systems calls and sends them on to DOS when it needs to.

If you are using TopView on a floppy system, for every application you are running you need a set of data files and programs to go with it. You would very probably lose your Ramdisk if you were working with TopView. A hard disk system is much more feasible. It is no fun to give up your Ramdisk. You should be aware, however, that you may not need your Ramdisk as much with a multi-tasking environment like TopView. A programmer might have the Personal Editor in one partition doing editing, a second partition running the compiler, and the third one testing the application. Once you load PE into memory you don't need to reload it from disk anymore.

PJ: It might be possible to organize your floppies to maximize the benefit of using TopView. Instead of having both the programs and data files on one diskette you could separate them so that you could load all of your software from one floppy and have the data/text files on another.

Any other comments on TopView?

RON: It doesn't currently handle pipes/redirection or batch files. If you want to avoid looking at your I/O right away, you can send it to a file. This is not implemented now.

You do have to set up a TopView interface or driver for the applications you run under it. You tell TopView which program you wish it to control and TopView creates an interface file that it uses to handle everything. I didn't see an A: prompt at any time during the demo at Comdex. I'm not sure whether you ever actually end up in DOS if you are in TopView. It will be interesting to see what residual effect the TopView environment will have on products like Sidekick. I think these combi-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

nation programs may eventually be separated so you can run either a notepad, calculator, or clock as a distinct piece, rather than loading all of them in with the overhead they require.

With the active windows of TopView, you have several RESIDENT programs, even though the programmer wrote them as normal, load-from-disk software. We can skip the stay-resident code unless our application needs to filter keyboard input.

PJ: How many buildings did Comdex have for displays?

RON: There were six major locations. The main one was the convention center. I went in there the first day and spent from 10:00 am to 5:00 pm and didn't see it all. I did take 20 minutes for lunch. It is as big as they say. I just walked through and spent several minutes at the booths I wanted more information from.

The entrance to the main hall is worth mentioning. You walk in the front door and you run right into the AT&T booth. You have to walk through it or around it to continue your explorations. IBM was back a few booths, as well as Tandy, Alpha Micro, Zilog and so on.

PJ: What was the registration fee?

RON: Around $25. Probably half of the attendees were "guests" of various exhibitors. If you do plan on going, it is better to plan four months in advance, even registration. One especially nice feature about the convention was the ID tags. It is essentially a plastic credit card with your name, company, and address on it. When you go to a booth and ask for additional information, you hand them the card and they run it through a charge card machine that imprints the information onto a mailing label. Very efficient for both of you.

THE PROGRAMMER'S SHOP™

helps compare, evaluate, find products. Straight answers for serious programmers.

SERVICES
- Programmer's Referral List
- Compare Products
- Help find a Publisher
- Evaluation Literature free
- Dealer's Inquire
- Newsletter
- Rush Order
- Over 300 products
- BULLETIN BOARD - 7 PM to 7 AM 617-826-4086

Free Literature - Compare Products

Evaluate products Compare competitors. Learn about new alternatives. One *free call* brings information on just about any programming need. Ask for any *"Packet"* or "Addon Packet": ☐ ADA, Modula ☐ "AI" ☐ BASIC ☐ "C" ☐ COBOL ☐ Editors ☐ FORTH ☐ FORTRAN ☐ PASCAL ☐ UNIX/PC or ☐ Debuggers, Linkers, etc.

RECENT DISCOVERIES

Introducing-C. C Interpreter and training system. Nice. Thorough. PCDOS. Only $95

"C" LANGUAGE

	OUR PRICE
MSDOS: C86-8087, reliable	call
Instant C - Inter., fast, full	500
Lattice 2.1 - improved	call
Microsoft C 2.x	329
Williams - NEW, debugger	call
CPM80: Ecosoft C-now, solid, full	225
BDS C - solid value	125
MACINTOSH: Softworks	365
Megamax-object, full	295

Compare, evaluate, consider other Cs

BASIC

	ENVIRONMENT	
Active Trace-debug	86/80	75
BASCOM-86 - MicroSoft	8086	279
BASIC Dev't System	PCDOS	115
BASICA Compiler - BetterBASIC - 640K	PCDOS	185
CB-86 - DRI	CPM86	439
Prof. BASIC Compiler	PCDOS	89
SCREEN SCULPTOR	PCDOS	119

Ask about ISAM, other addons for BASIC

FEATURES

CEnglish - BBMS to C. Complete, powerful. MSDOS $795, dBase to C + $200.

GCLISP - "COMMON" LISP. Help, tutorial, co-routines, arrays, thorough. PCDOS $475.

EXSYS - Expert System building tool. Full memory. Probability. Why. Intriguing. PCDOS $295

EDITORS Programming

	RUNS ON	OUR PRICE
BRIEF - Intuitive, flexible	PCDOS	195
C Screen with source	86/80	75
EDIX - Nice interfaces	PCDOS	149
FINAL WORD-for manuals	86/80	215
MINCE-like EMACS	PC/80	149
PMATE-powerful	8086	195
VEDIT-full, liked	86/80	119

UNIX PC

COHERENT - for "C" users	PClike	475
COHERENT-NCI-Realtime	PClike	call
XENIX - plus C to MSDOS	PC	1295

Ask about run-times, applications, DOS compatibility, other alternatives. UNIX is a trademark of Bell Labs

LANGUAGE LIBRARIES

GRAPHICS: GraphiC-source	MSDOS	195
GRAPHMATIC-3D. FTN, PAS	PCDOS	125
HALO-fast, full-all lang	PCDOS	175
FILE MGNT: BTrieve-all lang	MSDOS	215
Cindex + -source, no royal	86/80	375
CTree-source, no royal	ALL	375
dbC ISAM by Lattice	8086	250
db VISTA-"Network"	MSDOS	465
PHACT-up under UNIX, addons	MSDOS	250
OTHER: CUtil by Essential	MSDOS	139
Greenleaf-200 + -	MSDOS	165
CSharp-**Real-Time**	MSDOS	600
PORTABLE C to PC, Mac, II	Many	125
SOFT Horizons-Communicate	PCDOS	139
SCREEN: CURSES by Lattice	PCDOS	125
CView-input, validate	PCDOS	195
MetaWINDOW-icons, clip	PCDOS	139
PANEL-many lang, term	MSDOS	265

FORTRAN

ENVIRONMENT		OUR PRICE
MS FORTRAN-86 - Impr	MSDOS	$ 239
DR Fortran-86 - full '77	8086	280
PolyFORTRAN-XREF, Xtract	PCDOS	165

OTHER PRODUCTS

Assembler & Tools-DRI	8086	159
Atron Debugger for Lattice	PCDOS	395
CODESMITH-86-debug	PCDOS	139
HS/FORTH-fast	PCDOS	239
IQ LISP-full 1000K RAM	PCDOS	call
MacASM-full, fast, tools	MAC	100
MBP Cobol-86-fast	8086	695
MicroPROLOG	PCDOS	285
Microsoft MASM-86	MSDOS	85
MSD Debugger	PCDOS	119
MultiLink-Multitasking	PCDOS	265
PC/FORTH + -well liked	MSDOS	219
Periscope-debug, "reset"	PCDOS	295
PFIX-86 Debugger	MSDOS	169
PL/1-86	8086	495
PLINK-86-overlays	8086	345
Polylibrarian-thorough	MSDOS	95
PolyMAKE	PCDOS	95
PROFILER-86-easier	MSDOS	125
PROFILER-flexible	MSDOS	125
Prolog-86-Learn, Experiment	MSDOS	125
SCIL-control "Versions"	86/80	349
SYMD debugger-symbols	PCDOS	119
TLC LISP-86-full, liked	MSDOS	250
TRACE86 debugger ASM	MSDOS	115

Note: All prices subject to change without notice. *Mention this ad. Some prices are specials.*
Ask about COD and **POs**. All formats available.

Call for a catalog, literature, and *solid value*

800-421-8006

THE PROGRAMMER'S SHOP™
128-J Rockland Street, Hanover, MA 02339
Visa Mass: 800-442-8070 or 617-826-7531 MasterCard

1207

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Now with 8087* Floating Point Support

The C Interpreter: *Instant-C*™

Programming in C has never been Faster. Learning C will never be Easier.

Instant-C™ is an optimizing interpreter for the C language that can make programming in C three or more times faster than using old-fashioned compilers and loaders. The interpreter environment makes C as easy to use and learn as Basic. Yet ***Instant-C***™ is 20 to 50 times faster than interpreted Basic. This new interactive development environment gives you:

Instant Editing. The full-screen editor is built into ***Instant-C***™ for immediate use. You don't wait for a separate editor program to start up.

Instant Error Correction. You can check syntax in the editor. Each error message is displayed on the screen with the cursor set to the trouble spot, ready for your correction. Errors are reported clearly, by the editor, and only one at a time.

Instant Execution. ***Instant-C***™ uses no assembler or loader. You can execute your program as soon as you finish editing.

Instant Testing. You can immediately execute any C statement or function, set variables, or evaluate expressions. Your results are displayed automatically.

Instant Symbolic Debugging. Watch execution by single statement stepping. Debugging features are built-in; you don't need to recompile or reload using special options.

Instant Loading. Directly generates .EXE or .CMD files at your request to create stand-alone versions of your programs.

Instant Floating Point. Uses 8087* co-processor if present.

Instant Compatibility. Follows K & R standards. Comprehensive standard library provided, with source code.

Instant Satisfaction. Get more done, faster, with better results. ***Instant-C***™ is available now, and works under PC-DOS, MS-DOS*, and CP/M-86*.

Find out how ***Instant-C***™ is changing the way that programming is done. ***Instant-C***™ is $500. Call or write for more information.

Rational
Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), 8087 (Intell Corp.), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

Moving?

Please send us both your old and new addresses, with Zip Codes. If possible, include the address label from a recent issue of *Programmer's Journal*. Allow three to four weeks for processing.

Programmer's Journal
P.O. Box 3000
Dept. EE
Denville, NJ 07834

You don't really need a car at Comdex. There are many buses to ferry you around. I was there for three days, but was totally burned out by the last day.

PJ: Were you looking for particular applications or hardware at Comdex?

RON: I was focusing on five areas. High-speed modems (2400 baud), multi-user systems which could be sold as single-user systems to start, database management products running under Unix based OS's, C compilers and which environments they were ported to, as well as other structured languages suitable for large applications.

PJ: What did you like about the high-speed modems?

RON: Several companies offered 2400 baud modems, for example Hayes and other mainframe companies. You could use them on regular telephone lines. They run around $900. Many will be available around the first of the year (1985). Some companies were showing products that would not be ready until September of 1985.

My other concern was Unix-based systems. Most of the machines that were multi-user were running Unix or Xenix. The applications on display were not low-end software products. They were more like Dental or Medical systems for larger, more expensive hardware. They had to be multi-user.

PJ: Did you see any MS-DOS emulators for Unix?

RON: I didn't see any. One booth, Venix I believe, had an MS-DOS flip method. You were either in Venix or MS-DOS. It didn't seem to be totally debugged yet. At some point it will be available.

PJ: What else impressed you?

RON: IBM's new Enhanced Graphics board was quite nice. Excellent resolution, wonderful character sets, and legibility approaching the monochrome display. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C — ITS USES AND ABUSES

by Rex Jaeschke, Editor of the C Journal

A discussion on the use of C as a commercial and applications programming language. (C) Rex Jaeschke, 1984. All rights reserved. Previously appeared in "data link" on 5/19/84.

There is no doubt that the ever increasing presence of C and UNIX has had and will continue to have a significant effect on software development. For a number of years these two phenomena were considered together, but now that is not necessarily the case. It is hard to imagine UNIX without C, but it is very easy to imagine C without UNIX. In fact, it may well be that the majority of C compilers installed today (including micro-computer versions) do not run on a UNIX-like system.

C has generated tremendous interest. It has its own cult-like following, disbelievers (most likely Pascal programmers), and mythology (C programs are portable). Now that C compilers are readily available for many hardware/operating system combinations (usually at reasonable prices), just where does C fit as a language? What kind of systems could benefit most and least by being written in C? Could there possibly be some weaknesses in this most sacred of languages? Can and should C be used for applications programming?

C is a mid-level or systems language. That is, C fits somewhere between assembly and high-level languages. On the one hand, it has many of the capabilities traditionally found only in assembly languages in that it allows the programmer to get quite close to the host machine architecture and instruction set using constructs such as pointers and auto-increment and decrement operators. Conditional compilation directives are supported, as are bit-field operators and address arithmetic. On the other hand, C supports high-level language constructs such as if-else, while, for and case statements.

C was designed by and for professional programmers. It is by no means a language for the beginner. Many of C's more powerful capabilities, particularly pointers and structures, require advanced programming experience before they can be fully utilized. Assembly language programmers will most likely take to C's terseness, power and unusual operators far more quickly than high-level language programmers will. C allows assembly language programmers to work at the "nuts and bolts" level without insulting them. BASIC, Fortran and COBOL programmers will find the lack of group move and compare capabilities primitive, along with the need to call a function to perform any nontrivial operation.

We have stated that C is a systems language, i.e., one that is used to write systems programs. For the purpose of this discussion, systems programs will include such tools as operating systems, language compilers and assemblers, file access methods (including DBMS), and possible communications software. Data entry, validation, update, report and number-crunching programs will be considered applications software, which uses the tools described above. It is unlikely that anyone knowledgeable in C would argue against its use in writing system software. However, considerably mixed feelings abound regarding the use of C in applications programming. It is this latter topic we will now address. One valid yardstick for evaluation of a product for a particular purpose involves finding out what the product can't do. While features are nice, shortcomings can be disastrous. We will take the negative approach and will play the Devil's advocate during this evaluation.

Applications programs typically involve several of the following: full-screen data entry, string manipulation, financial calculation, file processing and report generation. C is not particularly suited to any of these. Let's take a detailed look at each of these functions.

C does not support decimal arithmetic. Integer and/or floating point variables therefore must be used for financial applications. Floating point types give rounding errors while integers require scaling. printf and its related formatted output functions have no capability to perform commercial-type editing or scaling of such numbers beyond specifying column width and number of decimal places. To implement check-protect, floating currency, CR, DB and floating sign edit masks, the user must write the appropriate functions. COBOL, BASIC and even Fortran-77 are far superior at financial computation and display.

The reason that C can provide power similar to assembly language and maintain considerable machine independence at the same time is largely due to the fact that the C language has no data input and output statements. These capabilities must be provided by compiler vendor and/or user-written run-time library routines that are invoked from C programs. Although Kernighan and Ritchie of Bell Labs have made recommendations regarding the contents of a "standard" C run-time library in their definitive reference "The C Programming Language" (Prentice-Hall, 1978) the actual library implementation is left to the compiler vendor and/or programmer. Needless to say, there is considerable variation from one vendor library to the next. The most significant problem is with file I/O. The only standard- type support is character I/O. Indexed and direct access file support is quite often nonexistent and even sequential access methods vary. No particular cursor addressing mechanism is supported. Support for "get character without echo" as used in full-screen data entry is an implementation-dependent feature.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

As previously stated, C has no provision for group move and compare operations. Character strings are treated as arrays of characters. Comparing and copying such arrays requires a function call to a library routine. Although this works well and is easy to program, it is quite strange to high-level language programmers. Even Fortran-77 supports string manipulation. All strings are treated as variable length. While this is a nice feature, the programmer must determine the string's length by searching for its null terminator. No string descriptor prefix is used.

While the isolation of the possibly machine-dependent run-time library keeps the language definition small, the real work in an application will involve many library routines. Math, random number, sorting, color and graphics routines may or may not exist in any particular implementation. With the huge popularity of C and the myriad of compiler vendors, the C language is effectively in the public domain. There is no longer a central body that is able to effectively set standards. Unless this situation is rectified, it is difficult to imagine the evolution of a standard and powerful I/O mechanism or indeed a standard run-time library.

C variable names are limited to 8 characters, although some compilers recognize more. This is quite limiting, particularly to programmers experienced in COBOL. Even micro-processor assembly languages allow for 15-30 character names and labels.

C is an unambiguous language with only about 30 keywords. Due to the power of these basic statements and constructs and the fact that the programmer can effectively extend the language by using callable functions and still maintain portability, there is little need for language extensions. However, they do exist in some compilers and vary considerably from one vendor to another.

Despite the small number of keywords, there are a large number of operators, in fact more operators than keywords. The real challenge in writing C is to recognize the capabilities of each of these operators. C uses almost all punctuation characters on a keyboard. In fact, many keyboards in use today do not have all of the necessary characters to program in C. The less common ones used are ^ (bitwise exclusive OR), _ (underscore), [and] (block delimiters), ~ (one's complement), \ (backslash) and | (vertical bar). Many punctuation characters can be grouped to give a different meaning. For example:>,>=,>>, !, !=, =, ==, +, ++, &, &&, |, ||, ?, ?:, * (two different meanings) and assignment operators such as +=, /= and >>=. Concatenated operators and their order of evaluation can be quite confusing.

Programmers new to C may initially find programs hard to read as almost all code is written in lowercase. Lowercase and uppercase letters are treated by the compiler as distinct. In fact, C language keywords must be written in lower case.

C is weakly typed it permits the programmer to do seemingly silly things, particularly in mixed-mode arithmetic. This allows the programmer to do powerful tricks and also to make nasty errors. C code can be very tight and obscure to the point of being unreadable and therefore, unmaintainable. However, with care and discipline, a programmer can generate nicely formatted and easy-to-read code.

All floating point arithmetic in C is done in double precision. Although the extra precision may be nice, it is forced on the programmer in all cases. This somewhat contradicts the idea of a C cast whereby the programmer can explicitly interpret one data type as another; for example, (double) fvar and (single) dvar where fvar and dvar are single and double precision variables, respectively.

C uses stacks a great deal. Many functions use a large number of automatic variables, which are created on the stack each time their parent function is invoked at run-time. This feature effectively allows data overlaying to occur. Assume that a main program calls three subroutines, each of which uses variables local to that subroutine. As these local automatic variables are created at run-time and deleted when their

THE HAMMER

Software Tools in C

Designed for the programmer with the user in mind.

More than just BIOS/DOS access. , THE HAMMER ALSO provides screen attribute control, Smartmodem control, UNIX-like pattern matching, 123-like command menus, date & time processing routines, easy data entry & verification of dates, numbers, and strings, AND MORE. Super data entry routines give the programmer an easy to use, solid interface with the user.
Requires: IBM-PC/XT, DOS2, CI-C86 Compiler
Other C compilers supported soon.

Routines tried & true. $110 with source and manual.
For ordering or inquiring, call or write now:

O E S SYSTEMS
1906 Brushcliff Road
Pittsburgh, PA 15221
412/243-7365

Looking for the right tool for the job?
GET THE HAMMER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

parent function exits, they can reside in the same stack area. The effects and direct usage of stacks is usually foreign to high-level language programmers. The amount of stack space actually needed at run-time is impossible to calculate accurately as variables may or may not be created, depending on run-time conditions. Calculating the size stack necessary and handling stack overflow can be quite a problem. C does not provide support for using a variable number of arguments with function calls. No run-time error checking is performed on the number and type of real and dummy arguments. There are a number of other run-time error situations not handled well or at all by C. They include array subscript validity checking, floating point exceptions, and integer overflow and underflow. Typically, there is no compiler traceback feature to show the module where run-time errors occurred.

One major reason that software developers use C is the ease with which properly designed programs can be ported across different hardware and system software environments. With rapid development of newer and faster processors, software houses must somehow design programs that will have a life beyond the current systems. The time and cost of rewriting software for new target machines is too prohibitive if the vendor is to retain a reasonable market share.

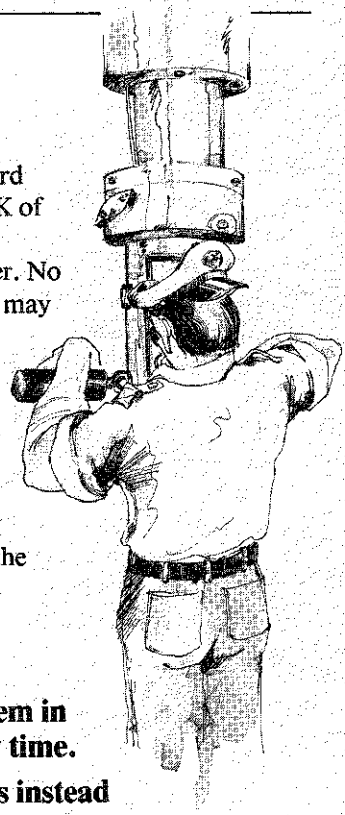
Using C does not in itself guarantee portability. It is very easy to write machine-dependent C code. The decision to make a program portable must be made before the program is designed. There are many different aspects of portability that must be reviewed. Designing portable programs takes care and discipline. Portability may mean running under different operating systems on the same hardware, the same operating system on different hardware, running on 8, 16 and 32-bit machines or on different versions of the same system. Portability must be defined before it can be claimed or achieved.

Before portability can be considered, the user must know what the language standard is. Herein lies a problem. Most C compiler manuals refer to Kernighan and Ritchie (K&R) for C language specifics, yet K&R does not completely define the language. Many idiosyncrasies are not obvious until the programmer has considerable experience with the language. Even when such quirks are found, the programmer is unsure if it's how C works or if it's how their particular C compiler works. Many programmers follow the UNIX implementation of C, but even then there are variations between UNIX and UNIX-like systems, and between Version 6, 7 and System III and V implementations.

Let's consider the problem of debugging. This necessary evil can eat up a considerable amount of development time, so debugging aids and tools are important. Very few compilers come with a symbolic debugger so the user is left to the old standard: compiling, testing, adding debug print statements, compiling, testing, etc., until it works and then removing or

Debugging Bugging You?

Torpedo program crashes and debugging delays with debugging dynamite for the IBM PC ...



UP PERISCOPE!

First, you install the hardware.

The hardware's a special memory board that fits in a PC expansion slot. Its 16K of write-protected memory contains Periscope's resident symbolic debugger. No runaway program, however berserk it may be, can touch this memory!

Then you UP PERISCOPE.

Use Periscope's push-button break-out switch to interrupt a running program ... even when the system's hung! Periscope supports Assembly, BASIC, C and Pascal. In addition to the usual debugging capabilities, some of Periscope's features are:

Stop your system in its tracks at any time.

Use symbol names instead of addresses.

Run a program on one monitor and debug on another.

Monitor your program's execution with Periscope's comprehensive breakpoints.

Debug memory-resident programs.

Put your time to better use.

The Periscope system is $295. It carries a 30-day money-back guarantee and includes the memory board, remote break-out switch, debugger software, 100-page manual, and quick-reference card. The memory board is warranted for one year. A demonstration disk is $5.00.

System requirements for Periscope are an IBM PC, XT or Compaq, PC-DOS, 64K RAM, 1 disk drive and an 80-column monitor. For MasterCard and Visa orders only, call 800/421-5300 (ext. R96) 24 hours a day. For additional information, call 404/256-3860 from 9 AM to 5 PM Eastern Time.

Get your programs up and running;

UP PERISCOPE!

Data Base Decisions / 14 Bonnie Lane /Atlanta, GA 30328

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

disabling the trace and finally compiling and using. Sound familiar?

The subtleties and minor differences between operators can take a long time to recognize. ((c=getchar()) != EOF) and ((c==getchar()) != EOF) can look the same or be misread to mean the same until you see the difference between = and ==. The compound statement delimiting braces and their scope can easily be misused and misread. Distinguishing between *ptr++ and (*ptr)++ can be difficult because, after all, don't the parentheses force the former meaning anyway? Not so in this case. The side effects of operators and the lack of order of evaluation in certain circumstances can also lead to confusion and unknown to the user, downright wrong results. Many C compiler packages don't even include a source cross-reference listing facility.

Despite all of the negative comments above, no doubt there are at least several responsible C compiler vendors whose product implements standard C (whatever that is) and who provide an extensive and somewhat portable run-time library. They may even provide good documentation, great support, no language extensions and a symbolic debugger. If such products exist, it is hard to identify them from the marketing and devotee hype.

Thus far, we have had some indication that C might not be the best language for applications programming. That does not mean that it can't be used as such. If you have the ego and the time, you could implement almost any application in almost any language. The point is, does it make sense? Unless a programmer really needs the unique capabilities provided by C, that person is unlikely to use it as a production language. C really isn't an alternative to BASIC, Fortran or COBOL. Possibly it's an alternative to Pascal and maybe assembler.

If you do choose to program a significantly large and/or complex system in C, here are some tips on doing it successfully.

Do:

- define portability requirements
- evaluate and buy several compilers
- use a compiler that at least follows the K&R library
- look for a compiler with a symbolic debugger
- understand the compiler's idiosyncrasies
- talk to software houses using C
- write run-time library tools in C rather than assembler
- write your own equivalent versions of library routines if you want them to really be portable
- change the name if you implement a customized version of a library routine
- ensure that your library has adequate file and terminal I/O support
- think about and plan for debugging before you start coding
- thoroughly understand the difference and implication of auto, static and extern variables and their scope
- take full advantage of the pre-processor
- adopt and enforce programming standards—be disciplined
- indent loop bodies to show their scope
- use temporary variables or parentheses if unsure of the order of evaluation of an expression
- be ready for some hair-tearing learning experiences.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Don't:

- use trainee programmers or programmers without significant C experience
- use language extensions
- skimp on comments
- write production code with a newly released compiler
- use labels and gotos
- interchange pointers and integer variables

One final comment, just in case you are totally disillusioned. Let us assume that you wish to program systems for use and/or sale on a specific hardware/software combination or on a series of very similar ones. In this case portability may not be an issue. With proper design you could develop a powerful set of C and assembler library routines. As your project progresses, so too will the number of tools in your software toolbox. Once the first project is implemented, those tools will form the basis for future systems. Implementing systems in C becomes easier, the more you have done. The whole key to this approach is good design and planning. Many of the principles required in good assembler programming can be adopted for use with C.

OK then. Even if all this is true, what about all those new computer scientists graduating with C experience? How will they affect the attitudes toward the usage of C? First, it will take some years for these people to get to a position where they will have an effect on the choice of programming language at all. Next, when you get to be a programming manager, strangely, many of your idealistic (and mostly theoretical) approaches to project management and program development prove to be impossible or undesirable. Choosing a language is but a small part of implementing a project. If a good project manager really understands C and the problems they are trying to solve, he is very close to deciding if C is the appropriate tool. After all, shouldn't we be picking the best solution for a problem rather than trying to justify the purchase of a particular compiler?

Like all other languages, C is not all things to all people, nor does it claim to be. For many applications COBOL, Fortran and BASIC will do the job just fine. Besides, what's so bad about COBOL anyway? Is it a bad language or are we just tired of it?

"To C or not to C. That can be a very difficult question." (My apologies to William Shakespeare.) □

Rollinsoft PCsoftware Don Buresh Ron Watson

PC FIRING LINE / PC UNDERGROUND™

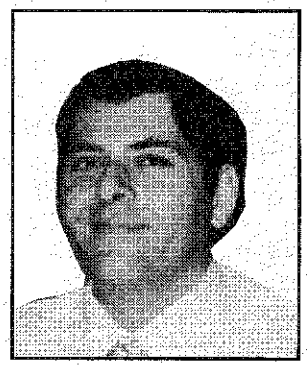
Technical Disk Magazine

Donald Buresh, C.D.P.

Throw away your paper magazines! Save shelf space, save programming time with the sizzling ready-to-use source code provided in each powerful bimonthly issue (currently 2 DS/DD disks per issue).

Donald Buresh, C.D.P., is our resident DOS expert. Don is a consultant in the Boston area and has authored FREECOPY, the INT 10H Video I/O routines for an IBM-PC compatible, and graphics routines for GWBASIC.

As editor of PCFL/PCUG (alias Bill Salkin), I defy you to find the following topics discussed in any magazine—be it paper or disk: the EXEC function, IOCTL functions, installing device drivers, critical-error handling, SOUNDEX codes, accessing the ROM-BIOS from LISP, a self-modifying BASIC function evaluator, "printf" source code, and flab reduction from IBM Pascal routines. High-sailing columns on Ada, Assembly, BASIC, C, FORTH, FORTRAN, LISP, Pascal, Hardware, and DOS. And NO paper magazine can supply ready-to-use source code, complete batch file libraries, window demos, animation demos, and utilities.



Bill Salkin

If you have been turned off by disk magazines, then look again. PCFL/PCUG—the magazine of the future—is available today for $12 per single issue (half the price of our competitors' and we contain at least twice their content) or $72 for a one-year (six-issue) subscription.

All funds in U.S. dollars. Foreign countries please send money orders and add $5 (U.S.) airmail for each issue.

Requires an IBM-PC, 128K RAM, and a double-sided disk drive. We ship ONLY DS/DD disks.

ABComputing

P.O. Box 5503, North Hollywood, CA 91616-5503 • (818) 509-9002

Guy M. Kelly Gary M. Rader Ken Holcombe Microcompatibles

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

USING THE C PREPROCESSOR

by Frank D. Greco, AGS Computers, Inc.

You C programmers out there know that typical programs written in the C language make extensive use of functions. These functions are quite similar to those modules of FORTRAN and Pascal also called "functions." The use of functions promotes the modularity of programming and thus facilitates the successful life cycle of any programming project (i.e., design, coding, maintenance). However, one of the disadvantages of using functions, especially on a microcomputer, is speed. Every time a function is called, the following occurs:

1. Current instruction address is pushed on the stack. Occasionally for the 8086-type machines BOTH the Code Segment Register and the Instruction Pointer are saved.

2. All the function arguments are pushed on the stack.

3. Local variables are created on the stack.

4. The function is executed.

5. Local variables are popped off the stack.

6. All the function arguments are popped from the stack.

7. The previous instruction address (again, sometimes both the CS and IP registers) is popped from the Stack and execution continues.

Now, for some applications that need every bit of speed the PC can muster, like computer simulations of real-world events (or out-of-this-world events, for that matter), some of your functions can be re-coded in assembler. (That has been my particular modus operandi lately: code mostly in C and use assembler if it is really necessary.) However, for maintenance purposes, it is often best to keep your code in a higher-level language. Let's try to get the best of both worlds by writing a program that avoids assembler and functions and uses macros instead. Let's look at a commonly needed function, creating a random number, and compare the differences between calling a random number function or use the underutilized C Preprocessor (usually the first pass of any C compiler) to create a random number macro.

Listing 1 shows a small Computer Innovations C86 program that calls a not particularly fast random number function given a predetermined seed. The algorithm is based on a common congruential method whose details are unimportant here.

Listing 1. Random number generation using function calls

```
#define MULT  25173 /* Numbers to be used in the Random */
#define INCR  13849 /*  Number Generator */
#define MAX   65536

unsigned int seed, mod;
int count[1000];

int
main(argc,argv)
int argc;
char **argv;
{
   int i, number;

   seed   = atoi(argv[1]);        /* 1st arg is seed */
   mod    = atoi(argv[2]);        /* 2nd arg is range */
   number = atoi(argv[3]);        /* 3rd arg is how many */

   get_time();                    /* Print Current Time */
   for (i = 0 ; i < number; i++)
          count[ rand(mod) ]++;   /* use FUNCTION call */
   get_time();
}
/*******************************************************/
int            /* Function to generate random numbers */
rand(mod)      /* from 0 to 65536    */
int mod;
{
   seed = (MULT * seed + INCR) % MAX;
   return (seed % mod);
}
/*******************************************************/
int
get_time()   /* This function uses some C86 particulars */
{
   struct regval{
          int ax, bx, cx, dx, si, di, ds, es;
   } srv, rrv;
   int seconds, tenth_seconds;

   srv.ax = 0x2C00;            /* DOS function 2C */
   sysint21(&srv, &rrv);              /* Call DOS */
   seconds = (rrv.dx >> 8) & 0xFF;
   tenth_seconds = rrv.dx & 0xFF;
   printf("Seconds: %d.%2d\n", seconds, tenth_seconds);
}
```

After compiling our program using the new CI-C86 Optimizing C86 V2.1 compiler, we run our program (called "rantest.c") as follows:

```
A> rantest 12345 10 1000
```

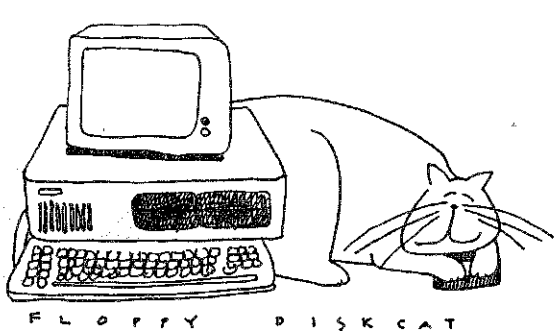
I received average times of about 1.50 seconds for generating 1000 random integers (including the calls to function get_time()). Now let's investigate using a macro instead of a function call.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Generating a similar set of 1000 random integers yields an average benchmark of about 1.40 seconds. That is approximately 7 percent faster. Since the C preprocessor textually replaces any macros that it finds in the source (i.e., a line that begins with #define) with its definition at compile time and NOT at run time, we get a slight increase in speed by avoiding function call overhead. Remember, the C preprocessor does not know anything about the C language. It is merely a macro preprocessor. Perkin Elmer's famous FORTRAN optimizer which vastly improves the speed of number crunching programs without any additional floating point hardware, performs its magic by expanding the existing FORTRAN subroutines in place similar to the way a macro preprocessor does and thus avoids FORTRAN subroutine overhead and subsequently gains additional speed.

I am not condoning writing function-less programs; that goes against my structured programming roots. What I am advocating is that an effective programmer should have a complete understanding of the available tools like a C preprocessor to be able to determine whether it is truly necessary to code a module of a program in assembler.

The C preprocessor is not a panacea. Sometimes you just cannot speed up a module by "macro-izing" it. In that case, if you need speed, cautiously consider assembler. For a production system on a PC that really needed the speed, probably assembler would be the way to go for a random number routine; for a VAX or similar minicomputer, I'd stick to using macros.And also note that you cannot take the address of a macro. The address of a function is legal and occasionally useful, especially when using an array of pointers to functions to gain random access to desired functions.I know that a 7 percent increase is not a tremendous difference in speed, but think what happens when you create several function-like macros. Believe me, any little bit helps on a 4.77 Mhz 8088.



Listing 2. Random number generation using a Macro definition

```
#define MULT    25173
#define INCR    13849
#define MAX     65536

/* "Random" macro with argument */

#define RAND(mod)       ((seed = (MULT * seed + INCR) % MAX) % mod)

unsigned int seed, mod;
int count[1000];

main(argc,argv)
int argc;
char **argv;
{
        int i, number;

        seed   = atoi(argv[1]);         /* 1st arg is seed */
        mod    = atoi(argv[2]);         /* 2nd arg is range */
        number = atoi(argv[3]);         /* 3rd arg is how many */

        get_time();
        for (i = 0 ; i < number; i++)
                count[ RAND(mod) ]++;   /* use MACRO instead */
        get_time();
}
/*************************************************************/
int
get_time()        /* This function uses some C86 particulars */
{
        struct regval{
                int ax, bx, cx, dx, si, di, ds, es;
        } srv, rrv;
        int seconds, tenth_seconds;

        srv.ax = 0x2C00;        /* DOS function 2C */
        sysint21(&srv, &rrv);   /* Call DOS */
        seconds = (rrv.dx >> 8) & 0xFF;
        tenth_seconds = rrv.dx & 0xFF;
        printf("Seconds: %d.%2d\n", seconds, tenth_seconds); □
}
```


YOUR CODE MAY BE WASTING ITS TIME!

THE PROFILER™ CAN HELP . . .

- **Statistical Execution Profiler**
- **Works with any language**
- **Completely configurable**
- **Up to 16 partitions in RAM/ROM**
- **Time critical code optimization**
- **Abnormal code behavior tracking**
- **Graphic presentation of results**
- **Easy to use menu interface**

THE PROFILER is a software package which gives you, the programmer, a powerful tool for locating time consuming functions in your code and allows you to performance tune your program. With **THE PROFILER** you can determine where to optimize your code for maximum benefit, then measure the results of your efforts.

Using **THE PROFILER**, you can answer questions like:
Where is my program spending its time?
Why is my program so slow? What is it doing?
Is my progam I/O bound? CPU bound? Are data buffers large enough?
How much improvement did my changes make?

THE PROFILER is completely software based and consists of a system resident driver and a monitor program. The memory partitions can range from 1 byte to 1 megabyte in size and can be anywhere in the address space.

NO ADDITIONAL HARDWARE IS REQUIRED!

Requires an IBM PC or compatible system with a minimum 64k and one drive.

THE PROFILER is available for $125.00 **(New Low Price)** from DWB Associates or ask your software dealer. To order or for more information, call or write DWB Associates. VISA/MC accepted. Dealers welcome.

dwb Associates
P.O. Box 5777
Beaverton, OR 97006
(503)629-9645

IBM is a trademark of IBM Corp. MSDOS is a trademark of Microsoft Corp.
THE PROFILER is a trademark of DWB Associates.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MEMORY SEGMENTATION EXAMPLES

by Bill Salkin, Used with the permission of PC Firing Line/PC Underground

These example files consist of sample runs from the assembler, linker, and DEBUG modules. Their purpose is to acquaint you, on a practical level, with segments.

The last example in this file shows how to use DOS calls to setup your own buffer. It is icing on the cake and not related to memory segmentation.

```
;--------------------------------------------------------------------------
;
;
;
;       This example  program defines  a data  segment containing  a
;       string called MSG. The number  of bytes in MSG is  redefined
;       with each run of the program (see the commented values), and
;       the results are detailed.
;
;
;
;--------------------------------------------------------------------------
DATA    SEGMENT PARA


;--Assembled perfectly, but  took the linker  about 1 minute  to link!
;  This segment contains 65535 zeroes, nothing much to speak of, but a
;  lot of it!

MSG     DB    65535 dup(0)


;--Though  a  segment  can  hold  65536  bytes  of data, the assembler
;  apparently can not handle that amount.  In  the  following example,
;  the assembler reports:  error 29: "Division by 0 or overflow."

;MSG     DB    65536 dup(0)


;--This next  example assembles  without errors  or warnings.  (Recall
;  that "DW" means "define word" where a word occupies 2 bytes.)
;
;
;MSG     DW    60000 dup('XX') ;60000 * 2 = 120000 bytes in a segment. NO Way!
;
;  Linker complains, though, as shown in the following run.
;
;       E>link temp;
;
;       IBM Personal Computer Linker
;       Version 2.00 (C)Copyright IBM Corp 1981, 1982, 1983
;
;       Attempt to access data outside
;       of segment bounds
;       Input File: TEMP.OBJ(A) pos: 0004E Record Type: A2
;
;       E>
;
DATA    ENDS
;--------------------------------------------------------------------------


;--------------------------------------------------------------------------
STACK   SEGMENT PARA STACK 'STACK'

        DB    40H DUP('STCK')

STACK   ENDS
;--------------------------------------------------------------------------
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;--------------------------------------------------------------------------
CODE    SEGMENT PARA 'CODE'

ASSUME  CS:CODE, DS:DATA, SS:STACK


MESSAGE PROC  FAR
        PUSH  DS
        SUB   AX,AX
        PUSH  AX
        MOV   AX,DATA           ;establish addressability of data segment
        MOV   DS,AX

;-- RETURN TO DOS

        RET


MESSAGE ENDP
CODE    ENDS                    ;END OF Segment
;--------------------------------------------------------------------------

        END   MESSAGE



;
;
;
;       The example is used to illustrate memory segmentation.
;
;
;
;--------------------------------------------------------------------------
DATA    SEGMENT PARA

MSG     DB    'HELLO','$'

DATA    ENDS
;--------------------------------------------------------------------------


;--------------------------------------------------------------------------
STACK   SEGMENT PARA STACK 'STACK'

        DB    40H DUP('STCK')

STACK   ENDS
;--------------------------------------------------------------------------


;--------------------------------------------------------------------------
CODE  SEGMENT PARA 'CODE'

ASSUME  CS:CODE, DS:DATA, SS:STACK


MESSAGE PROC  FAR
        PUSH  DS
        SUB   AX,AX
        PUSH  AX
        MOV   AX,DATA           ;establish data segment addressability
        MOV   DS,AX

        MOV   AH,9              ;request PRINT STRING service
        MOV   DX,OFFSET MSG     ;make DS:DX -> MSG
        INT   21H               ;invoke PRINT STRING service


;-- RETURN TO DOS

        RET


MESSAGE ENDP
CODE    ENDS                    ;END OF Segment
;--------------------------------------------------------------------------

        END   MESSAGE
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
E>DEBUG msg.exe

-u
095F:0000 1E            PUSH    DS
095F:0001 2BC0          SUB     AX,AX
095F:0003 50            PUSH    AX
095F:0004 B86109        MOV     AX,0961
095F:0007 8ED8          MOV     DS,AX
095F:0009 B409          MOV     AH,09
095F:000B BA0000        MOV     DX,0000
095F:000E CD21          INT     21
095F:0010 CB            RETF

Nothing new here, or is there?

-t
AX=0000 BX=0000 CX=0180 DX=0000 SP=00FE BP=0000 SI=0000 DI=0000
DS=094F ES=094F SS=0962 CS=095F IP=0001  NV UP DI PL NZ NA PO NC
095F:0001 2BC0          SUB     AX,AX
```

Notice the value in the CS register (95F). This points to the code segment containing instructions corresponding to the original source code. At the far left-hand side of this instruction we find the address 095F:0001. The 95F agrees with the value in the CS register, as we would expect, since CS points to the code segment. The number 1 in 95F:0001 is the "offset", called the instruction pointer, of this instruction into the code segment. Verify that the IP register contains the value 1 in the above DEBUG display.

```
-t
AX=0000 BX=0000 CX=0180 DX=0000 SP=00FE BP=0000 SI=0000 DI=0000
DS=094F ES=094F SS=0962 CS=095F IP=0003  NV UP DI PL ZR NA PE NC
095F:0003 50            PUSH    AX
```

Notice that the address at the left-hand side of the DEBUG display is now 95F:0003. This is the next instruction. The value in CS is the same, as we are in the same code segment, but the value in the IP register has changed to 3. The IP value does not simply increment by 1 each time. In fact, IP increments by the length of the instruction executed, and this value varies with the particular instruction.

```
-t
AX=0000 BX=0000 CX=0180 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS=094F ES=094F SS=0962 CS=095F IP=0004  NV UP DI PL ZR NA PE NC
```

This moves the address of the segment named DATA (see the source code listing) into the AX register in preparation for moving that value into the DS register.

Let's dump memory, starting at that address, to see if "961" really does point to the segment named DATA. (Kind of a preview.)

```
-d 961:0
0961:0000 48 45 4C 4C 4F 24 00 00-00 00 00 00 00 00 00 00 HELLO$..........
0961:0010 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
0961:0020 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
```

Indeed it does. Furthermore, we see the string HELLO terminated by a dollar sign, and followed by the STACK segment.

This is similar in appearance to our original source code, and you should think of source code as a mental picture or template of your code in memory.

```
-t
AX=0961 BX=0000 CX=0180 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS=094F ES=094F SS=0962 CS=095F IP=0007  NV UP DI PL ZR NA PE NC
095F:0007 8ED8          MOV     DS,AX
```

Moving the value in AX into the DS register makes the segment named DATA addressable from the DS register, and we can now refer to DATA as our data segment.

```
-t
AX=0961 BX=0000 CX=0180 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS=0961 ES=094F SS=0962 CS=095F IP=0009  NV UP DI PL ZR NA PE NC
095F:0009 B409          MOV     AH,09
```

Note the value in DS.

Let's experiment with segmentation by dumping memory. As DS points to our data segment (961H), we should be able to view the data segment as follows:

```
-d ds:0
0961:0000 48 45 4C 4C 4F 24 00 00-00 00 00 00 00 00 00 00 HELLO$..........
0961:0010 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
0961:0020 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
```

Indeed. Recall that DS:0 points to the first byte in our data segment, DS:1 points to the next byte, etc. As we expected, the first byte is the character "H", the next byte "E", etc., just as conceived in the source code.

We could also use the following to view our data segment, as the DS register contains the value 961.

```
-d 961:0
0961:0000 48 45 4C 4C 4F 24 00 00-00 00 00 00 00 00 00 00 HELLO$..........
0961:0010 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
0961:0020 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
```

The point is that at run-time the address of the segment named DATA (9610H) is determined, but this value must be moved into the DS register. After the move has occurred, DS points to the start of our data segment and we may use the DSA register to operate on this segment.

Did you notice that I said the address of our DATA segment was 9610H, not 961H? Addresses in segment registers, and shown in DEBUG, are multiplied by 16 to determine the actual memory address.

Suppose we request DEBUG to dump memory, starting at 1000H, as shown below. How will DEBUG interpret our request? Do you think it will start dumping memory at actual address 10000H?

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
-d 1000
0961:1000 36 8B 46 FA 89 46 FE 8B-76 FE 03 76 0A 8B 44 FF 6.Fz.F~.v~.v..D.
0961:1010 32 E4 89 46 FC B3 7E FC-61 7C 0F 83 7E FC 7A 7F 2d.F|.~|a|..~|z.
0961:1020 09 8B 46 FC 05 E0 FF 88-44 FF 8B 46 FE 40 89 46 ..F|.`..D..F~@.F
0961:1030 FE 48 3B 46 F8 75 D0 8B-E5 5D CA 08 00 55 8B EC ~H;FxuP.e]J..U.l
```

A quick glance to the right side shows gibberish. A quicker glance to the left reveals our number, 1000, but DEBUG has used it as an offset into the data segment (DS:1000). That is, DEBUG thought we wanted to dump memory at the actual address 9610H + 1000H, which is heaven knows where. It certainly is not in our data segment.

For more ordinary fun, try a few other values as an offset.

```
-d ds:1
0961:0001 45 4C 4C 4F 24 00 00-00 00 00 00 00 00 00 00    ELLO$...........
0961:0010 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
0961:0020 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
0961:0030 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
```

We have dumped our data segment, starting with its second byte.

```
-d ds:2
0961:0002 4C 4C 4F 24 00 00-00 00 00 00 00 00 00 00       LLO$...........
0961:0010 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
0961:0020 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
0961:0030 53 54 43 4B 53 54 43 4B-53 54 43 4B 53 54 43 4B STCKSTCKSTCKSTCK
```

We have dumped our data segment, starting with its third byte.

The next example program fails to establish addressability of a data segment. (The ASSUME directive for the assembler is present but the step that actually moves the address of the data segment into the DS register is commented out.) Therefore, whatever the value in DS is, it is regarded as pointing to our data segment! The DEBUG trace is interesting.

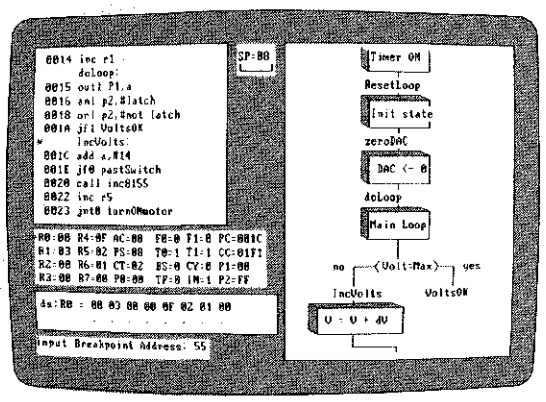
```
;-----------------------------------------------------------------------
CODE    SEGMENT PARA 'CODE'

ASSUME    CS:CODE, DS:DATA, SS:STACK


MESSAGE   PROC  FAR
          PUSH  DS
          SUB   AX,AX
          PUSH  AX
;;        MOV   AX,DATA          ;establish data segment addressability
;;        MOV   DS,AX

          MOV   AH,9             ;request PRINT STRING service
          MOV   DX,OFFSET MSG    ;make ds:dx -> MSG
          INT   21H              ;invoke PRINT STRING service


;--Return to DOS

          RET

MESSAGE   ENDP
CODE      ENDS                   ;END OF Segment
;-----------------------------------------------------------------------
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
E>DEBUG msg.exe

-u
095F:0000 1E            PUSH    DS
095F:0001 2BC0          SUB     AX,AX
095F:0003 50            PUSH    AX
095F:0004 B409          MOV     AH,09
095F:0006 BA0000        MOV     DX,0000
095F:0009 CD21          INT     21
095F:000B CB            RETF

-t
AX=0000 BX=0000 CX=0180 DX=0000 SP=00FE BP=0000  SI=0000  DI=0000
DS=094F ES=094F SS=0961  CS=095F  IP=0001  NV UP DI PL NZ NA PO NC
095F:0001 2BC0          SUB     AX,AX

          Notice that DS:0 points to 94F hex.

-t
AX=0000 BX=0000 CX=0180 DX=0000 SP=00FE BP=0000  SI=0000  DI=0000
DS=094F ES=094F SS=0961  CS=095F  IP=0003  NV UP DI PL ZR NA PE NC
095F:0003 50            PUSH    AX

-t
AX=0000 BX=0000 CX=0180 DX=0000 SP=00FC BP=0000  SI=0000  DI=0000
DS=094F ES=094F SS=0961  CS=095F  IP=0004  NV UP DI PL ZR NA PE NC
095F:0004 B409          MOV     AH,09

-t
AX=0900 BX=0000 CX=0180 DX=0000 SP=00FC BP=0000  SI=0000  DI=0000
DS=094F ES=094F SS=0961  CS=095F  IP=0006  NV UP DI PL ZR NA PE NC
095F:0006 BA0000        MOV     DX,0000
```

The source code statement "MOV DX, OFFSET MSG" has been converted to "MOV DX,0" in the unassembly, above.

Simulator/Debugger for IBM-PC for 8048 - 8051 - 7000 µCs



Execute and debug code for popular single chip microcomputers on your IBM PC. Dynamic display with windows for source code, control flow, registers, flags, memory, commands, and more! Set breakpoints, traps, etc. Cross-assembler & EPROM programmers, too.

	8048	8051	7000	Z8	
X-Assembler	✓	✓	✓	✓	$195
EPROM Prog	✓	✓	✓	✓	$245
Simulator	✓	✓	✓		$395

Demo disk and manual for simulator only $39.50

Cybernetic Micro Systems
Box 3000 • San Gregorio, California 94074 U.S.A.
(415) 726-3000 • Telex 171-135 Attn: Cybernetics

This indicates that relative to the segment in which MSG was defined (the segment named DATA), MSG has an offset of 0 bytes, or that MSG starts at the beginning of DATA.

When we dump memory, starting at DS:0, the value in DS is treated as a pointer to a data segment. We know this value does not point to the data segment in our program, but let's see what it does point to.

```
-d ds:0
094F:0000 CD 20 00 40 00 9A F0 FF-0D F0 42 02 4A 06 70 02 M .@..p..pB.J.p.
094F:0010 4A 06 E2 04 7E 05 4A 06-03 04 01 00 02 FF FF FF J.b. .J.........
094F:0020 FF FF FF FF FF FF FF FF-FF FF FF FF 4C 09 CA 2A ............L.J*
```

Pretty weird stuff. When a module is invoked, the DS register contains the address of an area called the Program Segment Prefix. We will discuss this area in a later article.

The moral is that your program must point the DS register to the data segment you wish to use. Otherwise, unpredictable (well, predictably bad) results will occur.

```
-u
095F:0004 B409          MOV     AH,09
095F:0006 BA0000        MOV     DX,0000
095F:0009 CD21          INT     21
095F:000B CB            RETF
095F:00^C
```

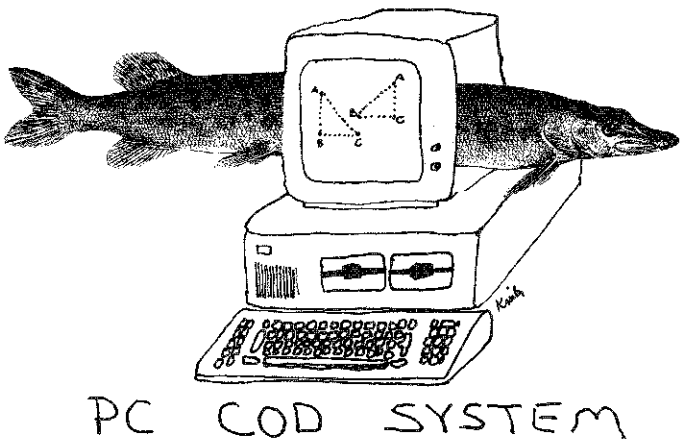
Just to tempt fate, let's print the string pointed to by DS:DX.

```
-g 95f:b
....
```

One moment please... technical difficulties experienced.

Unfortunately, some of the characters in the dump were printer-control characters, causing the dump to print as garbage, my printer to beep, go into expanded mode...

There is no protection in assembly language, no place to hide. A simple error can produce the most unexpected results.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;
;       This routine creates a keyboard buffer to receive
;       user-typed characters (max.) and prints the contents of the
;       buffer when the Enter key is pressed.
;
;
;
;       When setting up a buffer, the number of elements we want the
;       buffer to hold is placed in the first byte of the buffer.
;       The next byte is reserved for the number of characters the
;       user actually keys (filled in by DOS).
;
;       The last user-typed character in the buffer is followed by a
;       carriage return (which is considered to be part of the
;       buffer length), so an 8-character buffer has room for only 7
;       user-typed characters.  This is shown in the following
;       diagram:
;
;
;          2 bytes                    8 bytes
;
;        --Preamble--  ----------User-type characters----------
;
;        ------------  ---------------------------------------- -----
;        | 8 |  IN |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | CR | | $
;        ------------  ---------------------------------------- -----
;
;
;       Suppose that we want BUFFER to hold 7 user-typed characters
;       and define it by:
;
;
;            BUFFER   DB  buffer_length DUP('B'), '$'
;
;
;       where buffer_length = 8.  If the user keys too many
;       characters, the "$" may be overwritten because buffer_length
;       does not account for the "Preamble" as shown in the above
;       diagram.  (And if the "$" terminator is overwritten, how
;       does the print-string operation know when to terminate?
;       ANSWER: It doesn't, and will continue printing the contents
;       of memory until a "$" is encountered.)
;
;       The proper statement is:
;
;
;            BUFFER   DB  (2 + buffer_length) DUP('B'), '$'
;
;       This means to create an area containing (2 + buffer_length)
;       bytes, where each byte is the character "B".
;
;-------------------------------------------------------------------

buffer_length  equ  8          ;define buffer_length to = 8

;-------------------------------------------------------------------
STACK    SEGMENT PARA STACK 'STACK'

         DB  40H  dup('stck')
STACK    ENDS
;-------------------------------------------------------------------

;-------------------------------------------------------------------
DATA     SEGMENT PARA

MSG      DB  10,13,'Enter Number',10,13,'$'
CRLF     DB  10,13,'$'       ;carriage return/linefeed
BUFFER   DB  (2 + buffer_length) DUP('B'), '$'

DATA     ENDS
;-------------------------------------------------------------------



;-------------------------------------------------------------------
CODE   SEGMENT PARA 'CODE'

ASSUME   CS:CODE, DS:DATA, SS:STACK


BUFF     PROC  FAR              ; STANDARD LINKAGE TO DOS
         PUSH  DS               ;  DO
         MOV   AX,0             ;
         PUSH  AX               ;           NOT
         MOV   AX,DATA          ;
         MOV   DS,AX            ;                   ALTER!


START:   MOV   AH,9             ;request PRINT STRING service
         MOV   DX,OFFSET MSG    ;make DS:DX -> MSG
         INT   21H              ;invoke PRINT STRING service


;--Read (buffer_length -1)  characters into BUFFER

         MOV   BUFFER,buffer_length    ;move length into first byte
         MOV   AH,0AH           ;request BUFFERED KEYBOARD INPUT service
         MOV   DX,OFFSET BUFFER         ;make DS:DX ->BUFFER
         INT   21H              ;invoke BUFFERED KEYBOARD INPUT service

;--Move cursor to start of next line

         MOV   AH,09H           ;request PRINT STRING service
         MOV   DX,OFFSET CRLF   ;make DS:DX -> CRLF
         INT   21H              ;invoke PRINT STRING service

;--Print contents of BUFFER

         MOV   AH,09H           ;request PRINT STRING service
         MOV   DX,OFFSET BUFFER         ;make DS:DX -> BUFFER
         INT   21H              ;invoke PRINT STRING service

;--Go get next number (an infinite loop)

         JMP   START            ;Goto START


BUFF     ENDP                   ;END OF PROCedure
CODE     ENDS                   ;END OF Segment
;-------------------------------------------------------------------


         END   BUFF             ;EXECUTE PROGRAM STARTING AT `BUFFER'
```

□

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

INDEX FILES PART I

by Ronald Moak

Sooner or later if you program long enough you will find the need to rapidly retrieve data from a large data file. If the file is a random access file and you know the record number of the information you want, there is no problem. All you need to do is insert the record number into the BASIC get record function and the data is ready to be processed.

However, in most cases we know some of the information stored in the record but have no idea where it is within the file. Or we may wish to retrieve data from the file differently depending upon what information we need. If the file is a parts file, we may want to retrieve the information based upon part number, or if we do not know the part number it would be nice to be able to enter the item description and have the program select the appropriate record from either one of the inputs.

If the file we are accessing is random and sorted, our job is a little easier. A good search routine should be able to find the record fairly quickly. On the other hand, if the file is sequential, we have no choice but to start at the beginning and search each record until we find the one we want. In these days of high- speed computers, long waits for the program to search through a couple thousand records to find each record is unacceptable.

Taking little care when setting up the files used by the program can save the user lots of time running the program. In this article we will cover the use of the index file as one method of rapid data retrieval. If you are familiar with any of the many database programs available for the IBM PC (from PC-FILE to dBase II), you have probably used index files.

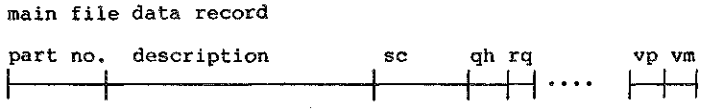
In practice the index file is a sub-file that contains information on the primary file. Index files are composed of two fields. The first is a subset of the information stored in the main data file, often referred to as a key. This subset is composed of one or more fields of data that have been concatenated together into a string. Numeric data is converted to string before being added to other string data. When the user enters the key information, the program first searches the index file for a match between the user-supplied values and the values stored in the key. When the key matches then the program uses record pointer (the second item stored in the index file) to locate the record stored in the main data file.

Index File			Data File		
Rec #	Key	Ptr	Rec #	Part No.	Description ... etc.
01	A100-7	14	01	A122-3	
02	A122-0	09	02	C115-0	
03	A122-1	08	03	C116-0	
04	A122-3	01	04	B111-9	
05	B111-8	05	05	B111-8	
06	B111-9	04	06	D100-6	
07	B119-1	10	07	H103-1	
08	C115-0	02	08	A122-1	
09	C116-0	03	09	A122-0	
10	C176-0	15	10	B119-1	
11	D100-6	06	11	D124-7	
12	D124-7	11	12	D125-1	
13	D125-1	12	13	E117-0	
14	E117-0	13	14	A100-7	
15	H103-1	07	15	C176-0	

The real advantage to index files is the speed with which they may be searched and sorted. Because the record lengths in index files are typically no more than 15 to 25 characters, they can be read far faster than data files with records lengths of one to two hundred characters. With a file buffer of 512 bytes, eight 25-byte records can be read into the buffer for each disk read. The shorter record lengths allow more records to be read into RAM, allowing a faster sorting of the file. By sorting the index file instead of the main data file, there is no chance of destroying the main data file if something should happen during the sorting process.

Let's take a look at a typical inventory problem and speed up record processing with the use of an index file. The main data file is composed of the following pieces of data:

```
1) part number              - 8  byte string
2) part description         - 40 byte string
3) supplier code            - 6  byte string
4) quantity on hand         - 2  byte integer
5) reorder quantity         - 2  byte integer
6) purchase price           - 8  byte double precision
7) selling price            - 8  byte double precision
8) volume discount percent  - 2  byte integer
9) volume discount mim qt   - 2  byte integer
```



If the part number used by this company was composed of sequential numbers, we could use the part number as the pointer to the record and our job would be done. Unfortunately, this company uses part numbers that contain both alpha and numeric characters (i.e., 145A-1). We will set up an index file that uses the part number as key. In operation the user will enter the part number and the record will be displayed on the screen for modification. The index file is composed of the following two data items:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

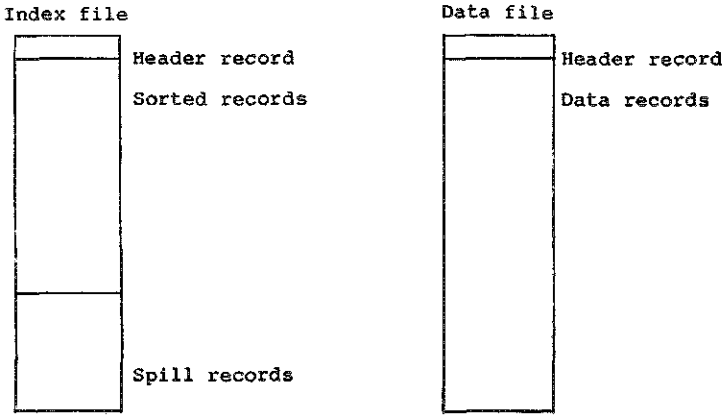
```
    1) part number      - 8 byte string
    2) record pointer   - 2 byte integers

index file data record

 part no. pointer
```

Our record pointer is a 2-byte integer and is limited to a database with 32767 records or less. If you are dealing with databases that are larger than that it is doubtful that the PC will handle them efficiently. BASIC limits the record number in GET and PUT statements to 2-byte integers.

So far in our example we have a data file composed of 9 items with a record length of 78 bytes and an index file with 2 items and a record length of 10 bytes. When designing index files it is important to consider the size of the record. In our example every 1000 records added to the database will add another 10K to the index file. For applications that strain the storage capabilities of the system, it may be necessary to reduce either the size of the data records or the index records.

Accessing data, updating, and sorting index files can range from being very complex to being fairly simple. A complex structure is exemplified by a program such as dBase II, where indexes may automatically be updated every time a record is added to the main file. In most cases new data is added to the primary file, then an indexing program is run on the file. The index file is then used for searching the file when updating records or reporting.

The structure we will cover in these two articles is intermediate in complexity. We will develop routines that automatically add records to the index file when they are added to the main file. The added records are placed in an overflow area at the end of the file. When the index file is searched, a binary search is performed on the main body of the file. If the record is not found in that area, then the overflow area is sequentially searched. The routines are designed so that when the number of records in the spill area exceeds a certain amount the index file is automatically sorted.



A number of routines are needed to handle the moving of data in and out of the database. These routines handle the following operations:

Open database
Add record
Find record
Read record
Write record
Delete record
Sort index file
Reindex file

The routines we will build require that each key is unique. If you are dealing with data files that do not have unique keys, you will need to make minor changes to the Find record routine to force it to find the first occurrence of the record.

The header record of each file contains information about the file that is needed for the routines to function correctly. The header record for the data file contains the number of records stored in the file. The header record for the index file contains the total number of records in the file and the number of sorted records.

The following routine will open the database and initialize the variables that contain the header information. The code presented in this article is designed to compile under IBM's BASIC compiler. The code should work under the interpreter by reformatting the code and adding line numbers.

(In Part II of this article we will explore the routines needed to move data into and out of the database using the index file.)

```
'*********************************************************
'*
'*   Open database and index files
'*
'*********************************************************
10000 open "INVENTOR.DAT" AS #1 len = 78
      open "INVENTOR.NDX" AS #2 len = 10

      field 1, 2 as FDAT.RECNUM$            'header record data file

      field 1, 8  as  FPART.NUMBER$                 _  'data file record
               40 as  FPART.DESCIPTION$             _
               6  as  FSUPPLIER.CODE$               _
               2  as  FQUANITY.ON.HAND$             _
               2  as  FREORDER.QUANITY$             _
               8  as  FPURCHASE.PRICE$              _
               8  as  FSELLING.PRICE$               _
               2  as  FVOLUME.DISCOUNT.PERCENT$     _
               2  as  FVOLUME.DISCOUNT.MIM.QT$


      field 2, 2 as FNDX.RECNUM$,     _   'header record index file
               2 as FNDX.SORTNUM$

      field 2, 8 as FXPART.NUMBER$    _   'data record index file
               2 as FXREC.NUMBER$

      get #1,1
      DAT.RECNUM = cvi(FDAT.RECNUM$)

      get #2,1
      NDX.RECNUM  = cvi(FNDX.RECNUM$)
      NDX.SORTNUM = cvi(FNDX.SORTNUM$)
      return
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

STRUCTURED PROGRAMMING LANGUAGE

An explanation, in the form of a sample program, of Dennis Baer's new language.

```
*CL

  Hello, My name is Dennis Baer and I would like to introduce you to
 my new programming language. It is called the Structured Programming
 Language and it runs on most MS-DOS computer systems. WAIT.....

*DL 10
*CL
*DS  It has the following statements, functions, and features.
*DL 2
*DS                    * IF THEN ELSE statements that can use BEGIN blocks
*DL 2
*DS                    * WHILE Loops
*DL 2
*DS                    * REPEAT Loops
*DL 2.
*DS                    * FOR loops with INTEGER or REAL indices
                  and fractional INCREMENTS.
*DL 2
*DS                    * Built-in Graphics statements such as:
                  PSET,LINE,CIRCLE,DRAW,PAINT,GET,PUT,etc.
*DL 2
*DS                    * Powerful STRING manipulation functions such as:
                  + (Concatenation), MID$,LEFT$,RIGHT$,ASC,LEN,
                  CHR$,etc.
*DL 2


*PC
*CL
*DS                    * INPUT/OUTPUT functions which allow RANDOM/SEQUENTIAL
                  access such as:
                  OPEN,CLOSE,INPUT,OUTPUT (with USING option),
                  FIELD,PUT,GET,LSET,RSET,WRITE,etc.
*DL 2
*DS                    * Variable declarations that are easy to read.
                  For example: REAL x,y,z;
                  STRING ARRAY String(10),Chars(50);
*DL 2
*DS                    * Mathematical functions such as:
                  ABS,SIN,COS,TAN,EXP,FIX,INT,LOG,SQR,RND,etc.
*DL 2
*DS                    * PROCEDURES, labels,variables with NAMES of up to 40
                  characters in length. For example: Display_the_line
*DL 2
*DS                    * CHAIN and automatic COMMON features.

*DL 2
*DS                    * REAL,INTEGER,STRING scalar and ARRAY types.

*PC
*CL
    The Structured Programming Language is implemented as a translator. The
 processor translates a source program in the language to a program in GW
 Microsoft BASIC for your particular MS-DOS machine. This program is then
 loaded into your Microsoft BASIC interpreter workspace and saved with the
 ASCII option. The program can then be COMPILED with a Microsoft GW BASIC
 compiler for your machine, or the generic MS-BASIC compiler. It is highly
 recommended that you use a COMPILER for BASIC. It will make your programs
 run much faster.

*PC
*CL

 Here is a sample source listing of a processed program MDM.SP.
 It is a modem program that sends a file on disk to the modem device.

 *** Structured Programming Language ***

  Date:10-22-1983  Time:18:17:54

  Source file:MDM.SP


    SP    BAS       Source Line

    1   1001   BEGIN
    2   1001
    3   1001          {This program outputs a file to the modem}
    4   1001
    5   1001          STRING File,        {Input file to be sent.}
    6   1001                 char,        {character string temporary.}
    7   1001                 record;      {This holds the input record.}
*PC
*CL
    8   1001
    9   1001          INTEGER count;      {Record counter.}
   10   1001
   11   1001          ONERRGOTO File_not_found; {If the file is not found then
   12   1002                                     this error will be trapped.}
   13   1002
   14   1002          Open_the_file:
   15   1003
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
  16   1003          INPUT( 'Please enter name of the file to be transferred:',
  17   1003                 File );
  18   1004
  19   1004          OPEN( File FOR INPUT AS #1 ); {The file has been opened.}
  20   1005

   SP    BAS       Source Line

  21   1005          ONERRGOTO Modem_file_error; {If any error with modem it
  22   1006                                       will be trapped here.}
  23   1006
*PC
*CL
  24   1006          OPEN( 'COM1:300,N,8' AS #2 );      {Open the modem device.}
  25   1007
  26   1007          {The Input file and the modem have been opened successfully
  27   1007           now read each record and output it to the modem until the
  28   1007           end of the file.}
  29   1007
  30   1007          ONERRGOTO Transmission_error;
  31   1008
  32   1008          count:=0; {No records read so far.}
  33   1009          WHILE NOT EOF(1) DO
  34   1010          BEGIN
  35   1010                 LINEIN(#1,record); {Read the record as 1 full line.}
  36   1011                 OUTPUT(#2,STRING$(15,0) @ record); {Write 15 nulls and
  37   1012                                                     then the record.}
  38   1012           '
  39   1012                 {Clear out the input buffer,only if characters there.}
*PC
*CL
  40   1012                 IF LOC(2)>0 THEN char:=INPUT$(LOC(2),#2);

   SP    BAS       Source Line

  41   1015
  42   1015           '     count:=count+1; {count one more record.}
  43   1019          END
  44   1021
  45   1021          CLOSE(#1,#2); {Close the input file and the modem.}
  46   1022          OUTPUT(); OUTPUT();
  47   1024          OUTPUT('The File ' @ File @ ' has been sent.'); OUTPUT();
  48   1026          OUTPUT(count @ ' record(s) have been read and sent.'); OUTPUT();
  49   1028          STOP; {Done, end of program execution.}
  50   1029
  51   1029          File_not_found:
  52   1030          IF ERR=53 THEN
  53   1032          BEGIN
  54   1032                 OUTPUT(); OUTPUT('File:' @ File @ ' not found.');
  55   1034                 OUTPUT(); CLOSE(); RESUME Open_the_file;
*PC
*CL
  56   1037          END
  57   1039
  58   1039          ELSE
  59   1039          BEGIN
  60   1039                 Other_type_error:

   SP    BAS       Source Line

  61   1040                 OUTPUT(); OUTPUT('Error number:' @ ERR); OUTPUT();
  62   1043                 CLOSE(); RESUME Terminate;
  63   1045                 Terminate: STOP; {Other type error,stop.}
  64   1047          END
  65   1048
  66   1048          Modem_file_error: {Error with opening the modem device.}
  67   1049
  68   1049          OUTPUT(); OUTPUT('A problem has arisen opening the modem device.');
  69   1051          GO TO Other_type_error; {Use the Other type error exit.}
  70   1052
  71   1052          Transmission_error: {Error in file input or modem output.}
*PC
*CL
  72   1053
  73   1053          OUTPUT(); OUTPUT('Error in Input file:' @ File @
  74   1054                                ' or with modem device.');
  75   1055          OUTPUT(); OUTPUT('Output Record:' @ record);
  76   1057          GO TO Other_type_error; {Use the Other type error exit.}
  77   1058    END

              *** END OF TRANSLATION PASS 1 ***
*PC
*CL
```

Before you decide to purchase this product, I recommend that you send for the Structured Programming Language manual that describes the language for the MS-DOS implementation. I feel that this product is something that you will find useful in writing your own systems and applications for business and science. The manual and sample applications written in the Structured Programming Language will be supplied to you for free if you send a 320K or 160K diskette, return postage, and mailer. Send them to

Dennis Baer
25 Miller Road
Farmingdale,N.Y. 11735

You may also call 516-694-5872 if you have any questions.

*PC □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A COMMERCIAL PROGRAMMER LOOKS AT BASIC ON THE IBM PC

by Bette Laswell

What, you may well ask, is a commercial programmer? Well, there are basically two types of programmers: the scientific/technical, who work on engineering and mathematical applications, and the commercial, who work on business and people-oriented applications. These two areas are mutually exclusive. The needs are different. The approach is different. The knowledge required is different. To be perfectly blunt, neither type of programmer should muck about in the area of the other; results are less than they should be. We are here to talk about the BASIC language on the IBM PC from the commercial programmer's view. Some background first.

When I started with IBM in 1958, the two "mass market" machines were the 650 and the 305 RAMAC. The 650's memory was 2000 words (of 10 characters and a sign) on a drum which constantly revolved under a bank of read/write heads. The programming language was SOAP (Symbolic Optimum Assembly Program); patching in machine language to optimize (which meant to make it less slow) was necessary. One of my first jobs was writing and installing programs to costout garbage collection for the City of Detroit. At that time and in that place, garbage collectors earned more than IBM Systems Service Representatives. Ah, humility!

The 305 RAMAC was the first commercial random-access machine, and if you had one, you put it in the window on display. When I was being trained on the 305, much emphasis was placed on the heat generated by the thousands of tubes behind the wall of covers. Turning off the machine did not cause an immediate shut down; fans continued to cool the tubes for several minutes before shutdown was complete. A power failure during operation was an emergency; if the covers were not rapidly removed, the thousands of tubes would overheat and burst. You know, of course, what happened the first time I got on the machine. There may still be people wondering why that frantic young lady in the 3-inch heels was racing around the display window tearing the covers off that huge grey machine all those years ago.

Then came the 1401. It could have up to 16,000 positions of binary-coded-decimal core memory, and we all wondered how you could possibly use that much. At this time, commercial and technical people used different machines to meet their different needs. The IBM 1620 was the technical machine out at the time; I learned all about it but, not being an engineer, never worked on it. The 1401 printer belted out 600 lines a minute, and that was impressive.

The fun, however, came from the fact that carriage control on the printer was done via a paper tape with holes punched in it. The program would say "Skip to 5," and the printer would advance paper and tape until a hole was found in channel 5 on the tape. On those occasions when the appropriate tape was forgotten, the skip instruction would cause an explosion of paper from the back of the printer. It was possible to empty an entire box of paper in a minute or two, and the operators became adept at reaching the "stop" button in milliseconds. (We were, praise be, no longer in the window.)

The 1440 introduced the first removable disk packs. They looked like cake covers for very large cakes and cost hundreds of dollars, but took random access a giant step forward. It was on a 1440 that a team of five of us developed the prototype system to computerize directory assistance for the Bell System.

Then came the 360. Along with this really fine machine came three really lousy decisions. The first was to combine commercial and technical needs in one "mass market" machine. The idea had appeal in the marketplace, but the execution was painful, at least to those of us in the commercial end. The first tape sort had no default options when it encountered an unreadable record. (You will recognize PC attributes here, also.) If you were sorting thousands of records, which on tape is not unusual, you could not opt to bypass an unreadable record or print out what was there for later handling — you were bounced!

A big sales point for the 360 was its compatibility with the 1400 series, which brings up the second lousy decision. 360 training would be concentrated on trainees so they would have nothing to "unlearn." I had 360 time one Friday afternoon to sort a tape file produced on a 1401. The plan was to return the tapes to the customer bright and early Monday morning, all sorted and ready to run; we would really dazzle them with compatibility! About two hours into the sort, the 360 executed an "unusual end of program"; it dumped me. I told the 360-trained trainee/operator to bypass the record; he gave me a look of pity. So I told him to print it out and continue. With the arrogance born of being a trainee, he carefully explained — with very precise enunciation — that I had come to "unusual end of program" and was off the machine, regardless of the hare-brained ideas I had to the contrary. He made a move to go home, it being 5:15 on a Friday afternoon and all; but there was no way I was going to the customer without sorted tapes on Monday, so we kept trying the sort until it finally worked.

The third lousy idea? To have the people in the branch offices learn the 360 by coming in an hour early on Wednesdays, without compensation, for instruction. In my office, without even a cup of coffee; and everybody in the office was already working well over 40 hours a week. Great was the grumbling!

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

But they fixed the sort, we all learned the 360, and I taught 360 COBOL to half the English-speaking world. Then, my health broke, and I took a few years off. When computers again became interesting (to me), third-party programming looked to be a good idea. My years with IBM had been in the era of "bundled" prices — the rental of the machine included education, field engineering service, and systems engineering help in designing and installing systems. That approach was long gone upon my return, and small and medium-sized businesses used independent contractors for their programming needs.

To start, I took the jobs nobody else wanted — fixing programs that didn't work. The first job was an RPG II/IBM System 34 accounts receivable system wherein none of the six reports balanced to each other or to the control, and none of the employees would cooperate. Talk about good times! But there was also the fun job on the IBM 5120 of aging a herd of cattle for tax-shelter accounting. Jobs were beginning to come in more readily, then we moved. A non-location dependent approach seemed to be the way to go, so I began writing home-use programs to sell on the mass market. The selling is another story for another time; here we'll talk about the programming.

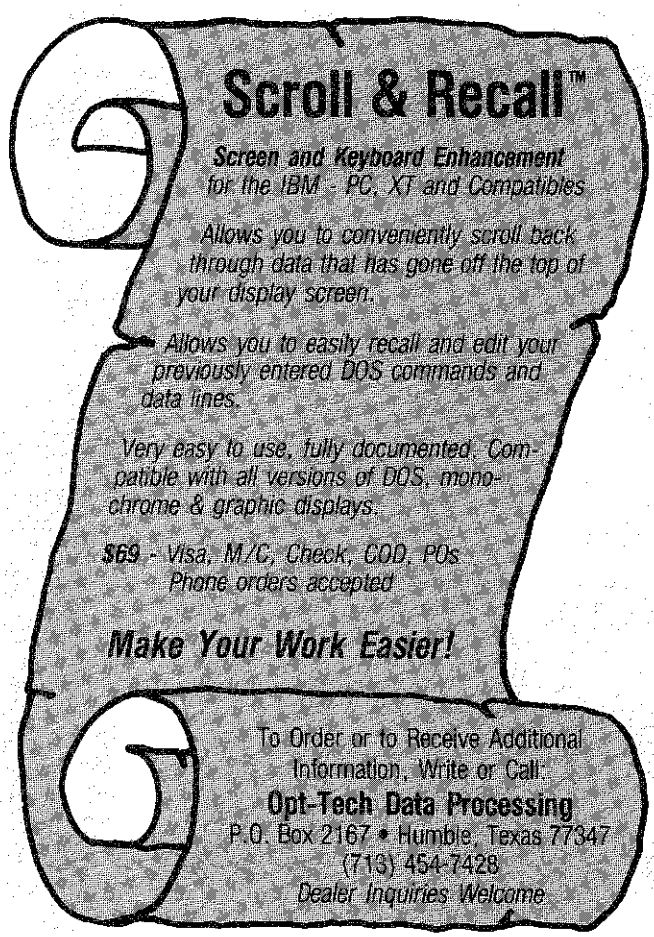
A first step in this new venture was, of course, deciding on a company identity. The choice made was to use my monogram BDL (Bette Dowdell Laswell) for three reasons. First, it would provide a quick reference to all the programs produced. Second, as the programs became known, it would stand for a high level of quality combined with an ease of accessibility. (You see, I come from a business background of three magic letters, only they stood for International Business Machines.) Third, and last, if by chance I stumbled onto a trademarked name, using my own initials would provide protection; I would have to change the company name, but I wouldn't get the socks sued off me. And since the programs would be for use in the home, Homeware was a logical choice — once my son suggested it.

BDL Homeware, then, is a series of programs for tasks necessary in running a home. They are written to require absolutely no computer knowledge or experience. It's tough programming. In business applications, the operator is told what to do, and fooling with "what happens if" is playing you-bet-your-job. On home computers, lots of people play "what happens if," so the program has to protect itself from being bombed by operators at play. My programs also have to protect themselves from unexpected responses from people who have never before seen a computer. The protection isn't the problem: it's just a lot of code. The problem is trying to divine where the program is vulnerable. BDL.ROAST, which calculates when to start roasting various meats and poultry, has to protect itself against those who would enter a negative when asked for the weight, or a huge or negative number when asked for the desired time of completion, for example. The really interactive programs have page after page of "protection" code. BDL.GIFT keeps track of personal characteristics of the people on your gift list, as well as who gave what to whom and when. Probably 15 to 20K is protection code.

Which bring us, at long last, to a commercial programmer's view of BASIC on the IBM PC. (You didn't think I was going to make it, did you?) Many problems would have been avoided if the folks who developed BASIC for the PC hadn't insisted on reinventing the wheel. BASIC on the IBM 5120 is pretty slick; it had been out for several years when PC BASIC was written. Oh, well, opportunities are lost all the time.

The major problem with BASIC on the IBM PC has to be the lack of I/O. (There goes the suggestion that I switch to PASCAL or C.) The overall quirkiness of this execution of the language grates on the nerves, too. So there, in a nutshell, it is. Let me explain.

Since the days of the IBM 305 RAMAC (announced in 1956), random access to records had been accomplished via "keys." Each record had a unique key, and the disk had a directory of keys. The program presented the key, and the I/O instruction found the correct record. The key could be a name, a number,


Scroll & Recall™
Screen and Keyboard Enhancement
for the IBM - PC, XT and Compatibles
Allows you to conveniently scroll back through data that has gone off the top of your display screen.
Allows you to easily recall and edit your previously entered DOS commands and data lines.
Very easy to use, fully documented. Compatible with all versions of DOS, monochrome & graphic displays.
$69 - Visa, M/C, Check, COD, POs
Phone orders accepted
Make Your Work Easier!
To Order or to Receive Additional Information, Write or Call:
Opt-Tech Data Processing
P.O. Box 2167 • Humble, Texas 77347
(713) 454-7428
Dealer Inquiries Welcome

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

whatever. Sorts were key sorts; the records stayed put, and only the keys were sorted. It saved a lot of time on sorts. Sometimes the keys were developed in the program by adding specified characters from name and/or address together; you would develop formulas to avoid duplicate keys. PC BASIC would go into cardiac arrest if a program tried to do math with non-numeric characters, and yet this is a good control in some applications. In PC BASIC you must know the physical number of the desired record to access it randomly. This is a major flaw for commercial applications.

Then there was the inscrutable decision to write a sequential file to the disk in a format different from a random file. It is desirable in many applications to access a file differently in different programs. Of course, it's possible to access a random file sequentially, but why, after nearly thirty years of random-access experience, is it necessary to fight the language to get a job done?

The cute little quirk that a disk field to be used numerically in the program must be defined with an extra position (in the Field statement) to avoid losing part of it seems unnecessary. At least, they could warn you about it! (I'm not even going to mention the reference manuals; I've never known one that wasn't at least semi-inscrutable.)

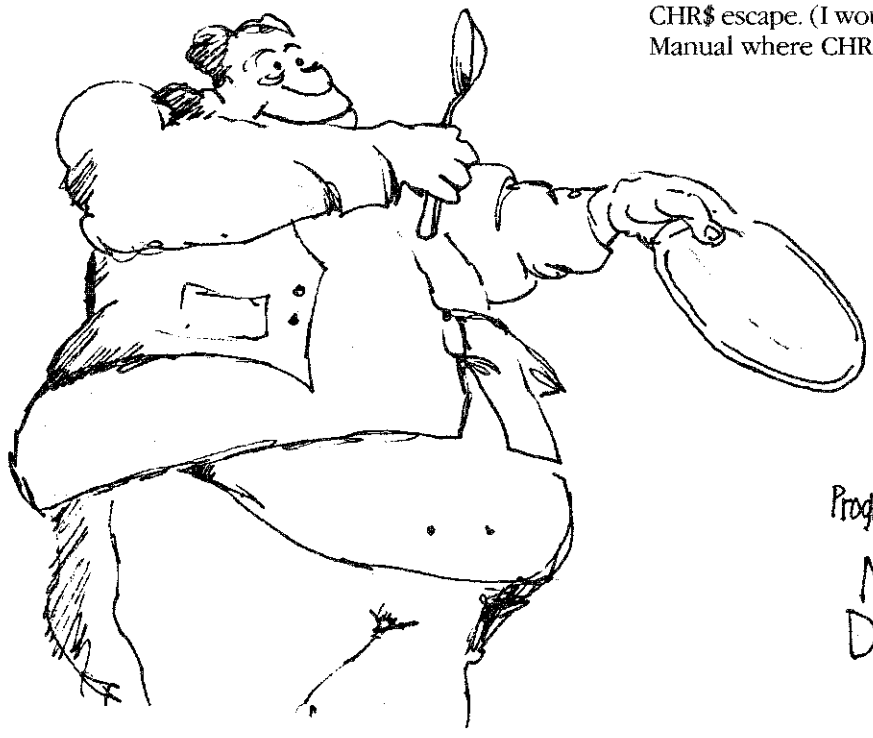
How about the lack of a logical compare for string data! A name read from the keyboard will never equal a name read from the disk unless both have completely filled the length of the field (as defined in the Field statement). "Jim" from the keyboard does not equal "Jim" form the disk unless it has been defined as a three-position field. The problem is easily overcome by a For/next statement, but why is that necessary?

Don't try to test a field from a random file for blanks using a compare. You can test several positions using the Left$ and get it to work, but not the whole field. It's written in ASCII, but testing for ASCII blanks doesn't work. Matter of fact, I don't know what works; I just work around it. Hours spent on such trivia make for crazy.

And whose idea was it to default to a 128-byte disk record when 512 gives better performance? In fact, not being able to specify a record length of greater than 128 without keying in a warning when BASIC is loaded is cumbersome. (I would also mention that the BASIC command which accomplishes this is not listed in the index, the table of contents, or the summary of BASIC commands, but I'm not going to complain about the manuals.)

But enough about disk; how about the print commands — or rather Lprint (mnemonics at their best!). There is, of course, the BASIC problem that certain decisions have been made about what type fields should have spaces before and/or after them. This problem can be overcome, but lining up numbers to print can be unnecessarily problematic. There is a Print/Lprint Using statement, to be sure, but each statement can have only one edit word and no spacing or tabbing. A simple line of print can require many statements to accomplish. Hours of writer's cramp could be avoided with a decent Print/Lprint Using statement.

When the need, or desire, to do anything other than straight printing arises, one realizes that BASIC doesn't support "fancies." You are exited to a machine-level approach of the CHR$ escape. (I would mention that the Technical Reference Manual where CHR$ is found is profoundly inscrutable, but



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

I'm not going to complain about manuals.) One thing you learn is that CHR$ statements sometimes cancel each other out, and the thing to do is rearrange the sequence until everything works. Another bit of knowledge acquired is that some CHR$ statements cause the next line of print to start in a position other than that specified, so throwing in occasional Print/Lprint statements to print nothing (but to get back in alignment) is necessary.

A real problem with hardware and not BASIC is that different printers interpret the CHR$ codes differently. A program that works well can fall apart with a change of printers. BDL.GROW keeps track of birth, growth, immunization, and doctor name and address information. A fun part of the program is that it will print a birth announcement with large print, overstrikes, emphasized print, the works — on an IBM dot-matrix printer or a Brand X dot-matrix printer that uses the same CHR$ codes. Another printer on the machine means the announcement isn't available. To get into printer specifications could be to defeat the intent of making the programs easy to use for people without computer backgrounds. And with a new printer being announced every five minutes, there would always be one that went belly-up on a CHR$ code. 'Tis a problem.

It would be nice if there were a code to indicate that automatic page overflow is desired in a given program. Along with the Width, Key, and other set-up statements, you could specify automatic overflow on a given number of lines, and the program would go to a specified subroutine when necessary. No more counting of lines. Heaven!

A problem for all programmers using BASIC, technical and commercial alike, are the so-called diagnostics. Each error message covers a minimum of 72 possible problems, it seems. The same error executed in different statement sequences can generate different error messages in each instance. Sometimes, it kills the program in one sequence and is ignored in another. I have not noticed a pattern in all this confusion. The problem that an error message about a subscript simply refers to anything in parentheses is true in many BASIC interpreters. Maybe this is an industry-wide contribution from the TRS 80, which is where I first encountered it. It might be said that listing the error messages in the manual in numeric sequence, but not putting the number with the error message on the screen is a problem. However, once you diligently search through the error messages to find the correct one, the explanation doesn't say anything specific enough to be of much value. So who cares? Get the statement number and puzzle over the listing; the errors BASIC finds are usually obvious, anyway. The subtle ones just slip by BASIC and wait to kill you with erratic execution.

So, is BASIC on the IBM PC absolutely terrible? Of course not. In conversations, in reading, and in limited experience with BASIC on other personal computers, I would say that it seems, in fact, to be the best available. Far and away. At least, just now.

Truth to tell, I'm looking forward to IBM coming out with their own PC BASIC. It takes Big Blue a very long time to get stuff out the door, but it's good when it gets there. From past experience you ought to be able to expect a less quirky interpreter and programming tools to please all programmers. And then all we'll have to worry about is the tendency of the computer to do just what we tell it, even when it leads to the never-never land of the endless loop.

By the way, there's another BDL program in release. It's BDL.CELEBRATE, and it keeps track of birthdays and other celebration dates. You tell it the month you're interested in, and it will give you all the people with "celebration" dates (birthdays, anniversaries, etc.) in that month — in order by date! It also does general mailings for any of up to five name and address files with labels, prints name and address "books" on request. Prints labels for the celebration people or for the general mailings, if you want. Thought you might like to know about all the BDL programs available right now. □

C

UTILITY LIBRARY

The C UTILITY LIBRARY is a set of 200+ functions designed specifically for the PC software developer. Use of the Library will speed up your development efforts and improve the quality of your work.

- BEST SCREEN HANDLING AVAILABLE
- WINDOW MANAGEMENT, COLOR GRAPHICS
- DOS 2 DIRECTORIES, COMMUNICATIONS
- KEYBOARD, PRINTER, TIME/DATE
- EXECUTE PROGRAMS, BATCH FILES
- STRINGS, BIOS, AND MUCH MORE
- ALL SOURCE INCLUDED—NO ROYALTIES

Available for Microsoft/Lattice $149, Computer Innovations $149, Mark Williams $149, DeSmet $99. Add $3 shipping. N.J. residents add 6% sales tax. Visa, MC, checks—10 days to clear.

Order direct or through your dealer. Dealer/Distributor inquiries welcome.

ESSENTIAL SOFTWARE, INC.
(914) 762-6605
P.O. Box 1003
Maplewood, N.J. 07040

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EXCERPTS FROM DTACK GROUNDED THE 68000/16081 SYSTEMS JOURNAL, Issue #31

by FNE, Copyright © 1984 by Digital Acoustics, Inc.

SURPRISING MEMORIES:

Let's face it, some folks are a lot smarter than some other folks. A strong contender for the title of 'World's Smartest Publisher' would be David Bunnell. You will recall that David asserted last year that Commodore was in trouble. Since that assertion came before, during, and after an unbroken succession of record quarterly sales and profits by Commodore and during a time when Commodore was growing at the rate of 2.3 times a year, compounded, David obviously knows something that the rest of us dummies don't. Proving that he possesses nearly limitless shrewdness, his editorial in May '84 PC World expresses *SURPRISE* that many PCs now have 256K of memory!

Let's see now: the runaway smash-hit winner this past year in the PC software arena has been Lotus' 1-2-3, no? And 1-2-3 really needs 256K to be really useful, yes? *Naturally, it should come as a surprise to a publisher of a PC-related magazine that lots of PCs now have 256K of memory!*

As the memory size in personal computers rises, a dilemma is becoming increasingly apparent. That dilemma is based on the fact that the Intel line of microprocessors are spectacularly ill-suited to work in memory spaces larger than 64K.

You will recall our repeated assertions that Microsoft's Bill Gates is no dummy when it comes to software. It has finally surfaced that Microsoft's continued refusal to expand its industry-standard BASIC into the new, greater-than-64K world is deliberate. *Microsoft will not produce an over-64K BASIC for the PC because it would be ungodly SLOW using those crippled Intel micros!* (May '84 *80-micro*, p.92).

Nearly two years ago (issue #11, p.7) we pointed out that memory has historically doubled every 18 months. If today's correct mass-market memory size is 256K then three years from now the correct size will be one megabyte. Just as today's cheap vanilla Intel micro is the 5MHz 8088, three years from now the cheap vanilla Intel micro will be the 8MHz 80186.

The 80186, a part which maintains architectural and instruction set compatibility with yesteryear's 8080, has very good performance inside 64K boundaries but is a miserable failure when attempting to work with larger memory spaces. (Incredibly, Intel's 32-bit iAPX 386, now under development, will continue the Intel tradition of 8080 compatibility.)

On the other hand, Motorola's cheap, vanilla micro three years from now will either be the 12.5MHz 68000 (available across the counter for two full years now) or something even *HIGHER* in performance.

Whether Motorola's cheap, vanilla micro three years from now is the 12.5MHz 68000 or something better, we can be assured that 1) that product will be software-compatible with the 68000, which has now been around for 3 1/2 years and is gathering software support and will *THEN* have been around for 6 1/2 years and will *THEN* have *MASSIVE* software support, and 2) that product will not be busted by being forced to be architecturally compatible with a (then) 13 year old 8-bit micro, and 3) that HALGOL will be able to work easily and naturally with memories in the megabyte range (we suspect this will be true of most other 68000 software as well, naturally).

Intel is today's clear and obvious leader precisely because it has maintained historical continuity with the 8080 architecture and instruction set and hence has a lot of software (developed for 64K memories) to back it up. Motorola is lagging today because it made a decision to break loose from compatibility with 8-bit micros and had to begin from point zero with respect to software support. (Motorola's original and incredibly naive plan to withhold the 68000 from the mass-computer marketplace exacerbated this situation for a while.)

In another three years the 68000 will have a solid, in-place software base and *NONE* of that software is going to be limited in any way by arbitrary memory restrictions or 8-bit compatibility. The folks with those megabyte Intel machines are going to be spending a lot of time worrying about "What *SEGMENT* am I innnn . . ." and worrying about the very substantial performance degradation associated with 64K segment limitations (will Microsoft *EVER* write a larger-than-64K BASIC for the Intel line of micros?).

It has been nearly a year now since we predicted an explosion of sales of 80186-based personal computers (#21, p.3). Well, Intel now has orders on hand for *TEN MILLION 80186's!* Since all those orders were (ahem!) *totally unexpected*, Intel

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

is only going to be able to produce *ONE* million 80186's this year, a shortfall equal to 90% of the demand.

Because Intel failed to foresee this huge demand for 80186's and hence failed to prepare adequate production facilities, the market window for those 80186-based products has narrowed substantially. The market window for the 80186 (and other microprocessors with 64K limitations) is going to slam shut when it becomes general knowledge in this industry that segmented micros are useless in one-megabyte single-user personal computers. And that is going to happen three years from now because in three years everybody is going to *HAVE* one megabyte in their personal computer.

As the old saying goes, the handwriting is on the the wall for those who can read it.

FNE MAKES MISTAKE:

Yes! Your FNE has (gasp!) made an error. We received this here anonymous phone call which opened up, "You made a serious error in your last newsletter. Microsoft's *Multiplan* for Mack is not either busted in C." But we checked that story three ways, we protested. We are *SURE* it is busted in C!

"No," the caller insisted, "it is not either busted in C. Multiplan for Mack is in fact busted *SQUARED* in *INTERPRETED* C! So is the Multiplan which runs on the Apple II, for that matter. You see," the caller explained, "if you really want to have a worthwhile program instead of just a toy, 128K just isn't enough memory to work with. The only way to sufficiently compress the code is to use an interpreter. For instance, Lotus was doing 1-2-3 for Mack in C, but Lotus *COMPILES* their C to native code. That's why Lotus has stopped working on 1-2-3 for Mack until Mack grows into 512K."

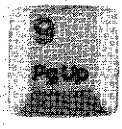
We guess we will have to take back what we have been telling our readers all along about our infallibility. Sigh. If you can't believe your FNE, who CAN you believe in?

ONCE UPON A MATH CHIP:

A newsletter editor once decided to have his two hardware type engineers develop a math processor peripheral using the Nat Semi 16081 for his 68000 attached processors. As you can read on pages 12 and 13 of his newsletter #22, a 'game plan' was devised toward this end. The (then) young project engineer (who has matured considerably since) read the 16081 spec sheet and noted that it was an asynchronous part. "I will interface the 16081 asynchronously!" exclaimed the (then) young project engineer.

The other project engineer, a fading dour pessimist, growled, "Spec sheet or no spec sheet I'll bet that part has to be interfaced synchronously. Let junior over there take his shot and then I'll show him how it's done!" Meanwhile, the newsletter editor planned to use the 16032 to drive the math chip, just in case.

So a prototpye was constructed, which is pictured on the front page of newsletter #24. It used an asynchronous clock, and worked correctly about 9,999 floating point operations out of 10,000. The (then) young project engineer improved the grounds, the +5 bypassing, he buffered the clock, he tried slower clocks. The prototype continued to work, 9,999 times out of 10,000. The fading dour pessimist suggested



ONE KEY IS WORTH A 1000 LINES

TALL SCREEN™ extends your screen at the touch of a key

PAGE BACK
RECALL
EDIT
FREEZE

PAGE BACK and **EDIT** up to 1000 lines of previous output. **RECALL** previous DOS commands or previous paths with a single keystroke. **EDIT** text anywhere on the screen. **FILE**, **PRINT**, or **FREEZE** windows on the screen.

Run programs unattended, review output at your convenience. Create customized keystroke profiles as in the IBM Personal Editor.

TALL SCREEN is a powerful tool for the PC user who values time and control.

$49.95 INTRODUCTORY PRICE

For orders or information see your local dealer or call **1-800-522-0290**

For the IBM PC®, PC Jr, XT, AT, or compatibles. Compatible with PCDOS 2.00, 2.10, 3.00 and 3.10. Compatible with most mouse systems and joysticks. Works with PROKEY™ and SIDEKICK™. Next day shipping on VISA/MC orders. Price includes all domestic shipping and handling. No extra charge for VISA/MC. MD & NM residents add sales tax.

QUALITAS™
QUALITY PROGRAMMING TOOLS

QUALITAS, INC., P.O. BOX 3AK UPB, LAS CRUCES, NM 88003 505-522-0290

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

using the same clock as the 68000, only divided in half. The system then worked 10,000 times out of 10,000 tries.

A second, somewhat improved prototype was constructed, and is pictured on the front of newletter #25. The two project engineers continued to learn more about the exact way the interface worked and which parameters were critical, but every attempt to use an asynchronous clock for the 16081 worked — 9,999 times out of 10,000. The newsletter editor reported the need for a synchronous clock on page 25 of issue 25. That issue also carried a schematic of a 16081/68000 interface.

You may find this hard to believe, but a photocopy of portions of newsletter #25 somehow found its way to Nat Semi headquarters in Silicon Valley. Here is an excerpt from a phone conversation which took place between the newsletter editor and someone at the Nat Semi plant in Nov. '83:

"You have an interesting writeup, but of course YOU. ARE. WRONG. about the synchronous clock." The newsletter editor tried to explain the many ways his project engineers had tried, unsuccessfully, to make the 16081 work asynchronously. But the guy at the other end talked a lot better than he listened. He explained, "WE. HAVE. THE. MOST. SOPHISTICATED. AND. EXPENSIVE. TEST. GEAR. THERE. IS! And, OUR. TEST. GEAR. PROVES. CONCLUSIVELY. THAT. THE. PART. IS. ASYNCHRONOUS." End of 'conversation.'

An excerpt from another phone call in Jan. '84: "You know, a couple of us here at Nat Semi now believe that the 16081 is really a synchronous part, not asynchronous. But we are having a heck of a time getting the chip boys to listen to us!"

An excerpt from a call to the Nat Semi headquarters in Mar. '84: "WE. ARE. TOO. BUSY. TO. SPEAK. TO. YOU. NOW. OUR. MOST. SOPHISTICATED. AND. EXPENSIVE. TEST. GEAR. IS. BUSY. RE-CHARACTERIZING. THE. 16081. AS. A. SYNCHRONOUS. DEVICE." Noooo! the newsletter editor answered . . .

Traditor!

The personal computer marketplace has only four significant participants at this time: Apple II/II+/IIe, Commodore 64, IBM PC and clones thereof and (collectively) the Z-80 CP/M machines. (Macintosh is not yet shipping in significant quantities.) What jumps right out at you is that *not one* of these four is even *remotely* state-of-the-art! Indeed, their performance is very 1977-ish (except for the very few PC programs which support the 8087). We believe that people sense this and are hoping for a White Knight, a modern machine which will lead them out of that antediluvian 1977 swamp.

In 1984 a state-of-the art personal computer will be based on the 68000 microprocessor (with a peripheral socket for the 16081, maybe). That was also true in 1983, of course, and what we got in 1983 was Fortune and LISA I. State of the art? Like hell! *Both were slower than an Apple II!* We have here a terrible disagreement between theory (68000 = state of the art) and practice (the Fortune and LISA I were dogs).

One of the things a newsletter writer does is *listen* (believe it or not). We ask other folks their opinions and then *we listen*. Even if we talked to someone else on the same subject an hour or a month ago, *we listen*. On the following subject we have found an amazing consensus. Here is that consensus:

First, a review: until quite recently, static RAMs were (all of them) incorrectly specified as a convenience to the specification writer. When one speaks of (for instance) a 150 nsec static RAM, one is speaking of a device which has a Taa (access time from address) of 150 nsec. Up until the past two years, that same part will feature (?) a Tcs (access time from chip select) equal to Taa, in this case 150 nsec. *THAT IS NOT IN ACCORDANCE WITH THE FACTS!* In fact, Tcs is *ALWAYS* much shorter than Taa. The reason is quite simple: There are many gate delays in the row and column address decoding, but very few gate delays in the chip select path. For example, Tcs of a Toshiba 150 nsec RAM is in fact 6 nsec +/− 5 nsec.

OPT-TECH SORT™

SORT/MERGE program for IBM-PC & XT

Now also sorts dBASE II files!

- Written in assembly language for **high performance** Example: 4,000 records of 128 bytes sorted to give key & pointer file in 30 seconds. **COMPARE!**
- Sort ascending or descending on up to nine fields
- Ten input files may be sorted or merged at one time
- Handles variable and fixed length records
- Supports all common data types
- Filesize limited only by your disk space
- Dynamically allocates memory and work files
- Output file can be full records, keys or pointers
- Can be run from keyboard or as a batch command
- Can be called as a subroutine to many languages
- Easy to use, includes on-line help feature
- Full documentation — sized like your PC manuals
- **$99** —VISA, M/C, Check, Money Order, COD, or PO

Quantity discounts and OEM licensing available

To order or to receive additional information write or call:

OPT-TECH DATA PROCESSING

P.O. Box 2167 Humble, Texas 77347

(713) 454-7428

Requires DOS, 64K and One Disk Drive

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A doctor, a construction engineer and a spec writer were having a drink one evening and got into a discussion of the world's oldest profession. "Well, on the seventh day God took a rib from Adam and created Eve. I guess that makes mine the oldest profession," asserted the doctor. "Nope!" replied the construction engineer. "On the first day God created the Heavens and the Earth out of Chaos. That makes mine the oldest profession!"

"Don't you guys know were all that Chaos came from?" the spec writer asked.

SHOVELING RIGHT ALONG:

Most of you undoubtedly believe that we shovel a little fertilizer (a natural by-product of dairy operations) from time to time (we did, after all, adopt the nom-de-plume *Felgercarb,* did we not?). From time to time, a whole bunch of you start shoveling in the opposite direction, and the past couple of months has been one of those times. About twenty of you are asserting that the high level language C is not really high level and hence has a vanishingly small time overhead as compared to assembly. Gentlepersons, that assertion is genuine Grade A cow-flop! A few facts:

Let us first observe FORTH, a language whose enthusiasts regularly claim has only 2-1 overhead. So Bruce Walker first introduces a standard FORTH for the Apple/DTACK system and then follows up with an Apple/DTACK FORTH which compiles nearly to native code with a concomitant 5-1 speed-up! Where did old 2-1 go? Out for a beer?

Now we come to the Mar '84 edition of *Computer Architecture News,* the ACM SIGARCH quarterly. In past issues, the performance of the 68000 and Z8000 were compared to the RISC I (a paper study where the RISC I is concerned, naturally). Assembled C code was originally used for the 68000 and the Z8000 and doggone if the RISC did not prove nearly 4 times faster than the 68000!

In the Mar '84 issue the benchmark tests are repeated, this time using assembly. *The 68000 sped up by a factor of MORE than six, and is now 1.5 times faster than the RISC I in four out of five tests!* In other words, C has a 6-1 overhead when used with the 68000 (the ratio was about 5-1 for the Z8000)! (Incidentally, the latest test was done with a 10MHz 68010. A 12.5MHz 68000 would skin the RISC I alive! *A 4-1 faster RISC I with the 68000 running C, indeed!)*

Use ALL the Power of Your MS-DOS, IBM PC-DOS, or CP/M-80 System with UNIX-Style Carousel Tools

```
ch "CP/M" "MS-DOS" <doc >newdoc
diff newdoc doc | more
ed newdoc
kwic newdoc | sortmrg | uniq | unrot >index
make -f makdoc ndx
```

CAROUSEL TOOLS are a proven set of over 50 programs designed to be used with pipes, redirected I/O and scripts. In the style of UNIX each Tool does one thing well, and the Tools can be used together to do more complex tasks.

YOU ACCOMPLISH MORE using Carousel Tools: better programming and documentation support, simpler data and file housekeeping, more general file handling.

TOOLS FOR PC/MS-DOS 2.x AND CP/M-80 are available now. The DOS ToolKit is $149. The CP/M ToolKit is $249 and includes a *shell* to provide pipes, redirected I/O, and scripts. Source code is available for $100 more.

ORDER YOUR TOOLKIT TODAY.

CALL OR WRITE:

CAROUSEL MICROTOOLS, INC.
609 Kearney Street, El Cerrito, CA 94530 (415) 528-1300

Carousel Tools and Carousel ToolKits are trademarks of Carousel MicroTools, Inc. CP/M is a trademark of Digital Research; IBM is a trademark of International Business Machines; MS is a trademark of MicroSoft; UNIX is a trademark of Bell Laboratories.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Gentlepersons, a *'more than 6-1 speed degradation'* can in no way be considered negligible. Will you please put down those shovels and accept the fact that C has the usual, expected and ordinary overhead associated with just about any high-level language? (PDP-11s excepted; C is largely stylized PDP-11 assembly code.)

So if you hire a new computer science graduate, one who is not yet mentally locked into the 'correct' way of doing things, it costs the company twice his salary. The same thing applies to other employees who might work on HALGOL such as your FNE. With this in mind, we assure you that we are being completely accurate and truthful when we tell you that during May '84 the investment Digital Acoustics has made in HALGOL (over a 2 year period) will pass $100,000.

Cheap HALGOL is *NOT* — for us.

Remember, a 'hacker' originally described German furniture-makers as so inept and clumsy that they were said to 'hack' furniture out of logs with an ax! The meaning of 'hacker' has become inverted with time and is now commonly used as a term of respect for a knowledgeable computer type. — FNE

A CHANGE OF SUBJECT?

For some reason, this seems to be an opportune time to report the following: Since mailing the last newsletter, we have chatted with two other newsletter writers whose views on UNIX are opposed to ours, to say the least. Here is their composite view as we understand it:

When viewing the UNIX scene, one appropriately focuses on new product releases and new companies entering the UNIX marketplace. One ignores profitability or the vendor/customer ratio as unimportant. Even if particular companies fail, UNIX is the eternal constant. It is very important for UNIX newsletter writers to produce um, somewhat optimistic projections of future UNIX sales (such as Jean Yates' 1982 projection of 600,000 UNIX boxes to be sold in 1983) because otherwise one will not be welcomed and taken to lunch/dinner by the heads of the companies one visits.

(By golly, Jean Yates IS regarded as a more popular dining companion than your FNE by the heads of Fortune, Sun Microsystems, Charles River . . .)

(As someone who runs a company, we just happen to think that profitability and a concomitant absence of business failure is downright *IMPORTANT!* Naturally, anyone who thinks profitability is important is going to carefully assess [twos's on the end, Bob] the reasonableness of sales projections, even going so far as to produce report cards for past years' projections. A proper UNIX person will naturally remain aloof from such crass, mundane considerations.)

"I am probably one of the many who have pointed this out but where do you get that parameter 'Address valid to (bar) AS Low' to be 40 nsec? The only spec near that value is the FC to AS Low. There seems to be some confusion between spec 11 and 11A, or do you know something about the 68000 AS timing that you're not telling?" Chris A., Pittsfield MA

Well, Chris, we do indeed know something which we have not — up until now — told. On the other hand, you are actually the first to question that 40 nsec address to (bar) AS assertion of ours, which (as you correctly point out) cannot be confirmed in any Motorola literature. Clearly, an explanation is called for.

The operating systems and BASIC interpreters, etc., for the 6502s and Z-80s were done in assembly. The programmers cut corners, used special tricks and in general did everything they could to improve the performance of their 8-bit software because that was *NECESSARY.* Adequate performance could not otherwise be obtained. These programmers knew (or learned later, after their companies had prospered) that they were not programming in a proper manner.

But these same programmers naturally want to write their software properly — *IF THAT IS POSSIBLE!* Once one has a 68000, with lots of registers (16 ea. 32-bit registers just like a VAX or IBM 370, in fact) and, get this, *FAR MORE THAN ADEQUATE PERFORMANCE,* it then becomes possible to program *CORRECTLY.* Correctly means using a transportable high-level language such as C or PASCAL. And so the folks programming the Fortune and LISA I (and Microsoft's Macintosh line of software) wrote their programs correctly. Result? The correctly-programmed 68000 machines are all slower (!) than the 1977 machines, which did not have the benefit (?) of being programmed correctly.

Do the folks programming the 68000-based products realize what they are doing, performance-wise? Yes. But, they will earnestly explain, *it is much more important for programs to be written properly than that they be fast.* Then they excuse themselves, explaining that it is time to disembowel yet another computer on the alter of the golden calf *Transportability.*

And so, the restive mal-ease of the personal computer marketplace: buyers have a choice of obsolescent 1977-performance computers *OR* they can choose a *MODERN* 68000-based machine that is *EVEN SLOWER!* And those are the only choices available! A pretty box indeed.

This explains the surge of interest that Bill J. has noted in the Tandy 2000. The 2000 is based on the 80186 which is in turn

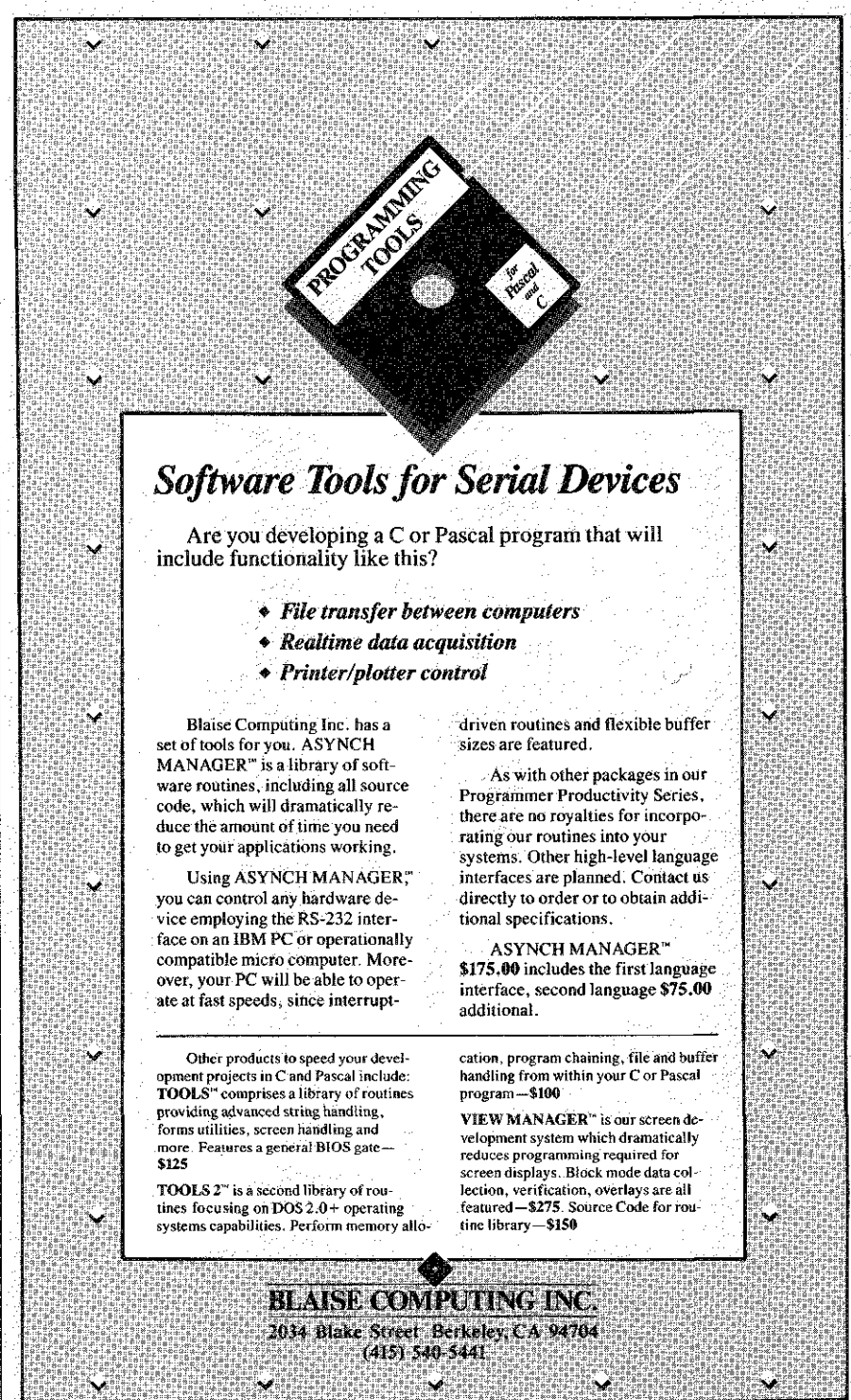
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

source-compatible (via a translation program) with the 8080A and source-compatible, period, with the 8088. So folks are turning to the Tandy 2000 in a desperate search for more modern performance even though the improvement is much less than 2-1 (vs an 8MHz 8086) or about 2.4-1 (vs a 4.77MHz 8088).

If only someone would put a gun to the head of those 68000 programmers and force them to write *BAD, FAST* code instead of *GOOD, SLOW* code then LISA would have been a success and Mack would, once production gets rolling, be enormously successful. As it is, it appears that Mack is attracting mouse fans and others who wish to travel north. Perhaps we are wrong, but we believe that the mass computer marketplace wishes to travel in other directions as well.

And perhaps we are wrong, but we think fast untransportable code is vastly superior to slow transportable code *IN THE MASS MARKETPLACE!* We are busily writing fast untransportable 68000 code (HALGOL) in an attempt to bring DTACK GROUNDED a step or three closer to the mass marketplace. Let us explain to Bill and to you other readers the full dimensions of our commitment:

In the electronics industry, it is well known that a small or medium-sized company has an overhead equal to its payroll (or double if the payroll is considered part of the overhead, as it usually is). In addition to the dollar figure that appears on the paycheck (before deductions) there are lots of hidden payroll taxes most employees are not aware of. Social security, unemployment, disability . . . Then there is the rent, the electricity and gas bills, the phone bill; liability, fire, theft and flood insurance. Employee health insurance. Property and inventory taxes. Business licenses. Supplies of all kinds. Computers and test equipment and software and technical publications. Furniture. Maintenance and waste disposal. All of these things, as is typical in this industry, almost exactly equal our payroll. □


PROGRAMMING TOOLS
for Pascal and C
Software Tools for Serial Devices
Are you developing a C or Pascal program that will include functionality like this?
♦ File transfer between computers
♦ Realtime data acquisition
♦ Printer/plotter control
Blaise Computing Inc. has a set of tools for you. ASYNCH MANAGER™ is a library of software routines, including all source code, which will dramatically reduce the amount of time you need to get your applications working.
Using ASYNCH MANAGER,™ you can control any hardware device employing the RS-232 interface on an IBM PC or operationally compatible micro computer. Moreover, your PC will be able to operate at fast speeds, since interrupt-driven routines and flexible buffer sizes are featured.
As with other packages in our Programmer Productivity Series, there are no royalties for incorporating our routines into your systems. Other high-level language interfaces are planned. Contact us directly to order or to obtain additional specifications.
ASYNCH MANAGER™ $175.00 includes the first language interface, second language $75.00 additional.
Other products to speed your development projects in C and Pascal include:
TOOLS™ comprises a library of routines providing advanced string handling, forms utilities, screen handling and more. Features a general BIOS gate—$125
TOOLS 2™ is a second library of routines focusing on DOS 2.0+ operating systems capabilities. Perform memory allocation, program chaining, file and buffer handling from within your C or Pascal program—$100
VIEW MANAGER™ is our screen development system which dramatically reduces programming required for screen displays. Block mode data collection, verification, overlays are all featured—$275. Source Code for routine library—$150
BLAISE COMPUTING INC.
2034 Blake Street Berkeley, CA 94704
(415) 540-5441

programmer's journal
The Resource Journal for IBM PC Programmers

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PERISCOPE, A SYMBOLIC DEBUGGER

Reviewed by Bill Salkin

Periscope
Data Base Decisions
14 Bonnie Lane
Atlanta, GA 30328
Price:$295

System Requirements:
IBM PC or XT, or AT
DOS 1.x, 2.x

What do you call a symbolic debugger that retains much of DEBUG's command language, yet enhances it; allows you to set code, memory, register, "sticky" breakpoints, single-step trace; leaps past interrupts (in a single bound); supplies a reset button, quick-reference card, alternate screens, hexadecimal arithmetic, hex. to dec. conversion; displays information in byte, word, or double-word format; compares one area of memory to another; loads files to and from memory; and supports the creation of customized display templates for the, until now, visually knuckle-busting areas like the PSP or FCB? What do you call this product?

"It's About Time"? or "Essential"? No — give it a reasonable name like Periscope and call the accompanying hardware Submarine.

The next several steps describe how to install Submarine and use Periscope.

STEP 1 (Install Submarine)

The hardware board, Submarine, contains 16K of memory and uses two consecutive I/O ports. If the default address of the memory or port settings used conflict with other attach ments in your PC, you must adjust the DIP switch settings on Submarine. This is not a difficult task, and the manual clearly describes how to set these switches.

STEP 2 (Install PS.COM)

The file PS.COM, provided on the distribution diskette, loads the resident debugger into the memory of Submarine. Installing the resident debugger is as simple as typing:

```
E   ps
```

STEP 3 (Program Preparation for Periscope)

The program shown below, called Dummy, displays "I've got the world on a string," using the DOS print-string function call. In the program, the procedure named Dummy and the label Print_string_service are declared as Public, for only Public items will be displayed under Periscope, using the original source code symbols.

```
;--------------------------------------------------------
Data    Segment Para

        STRINGY DB  "I've got the world on a string",'$'

Data    EndS
;--------------------------------------------------------

;--------------------------------------------------------
Stack   Segment Para Stack 'Stack'

        DB  40H DUP('STCK')

Stack   Ends
;--------------------------------------------------------

;--------------------------------------------------------
Code    Segment Para
Assume  CS:Code, DS:Data, SS:Stack

;--Standard Linkage to DOS

Public  Dummy, Print_string_service ;<******************

Dummy   Proc  Far           ;DO
        push  DS            ;
        sub   AX,AX         ;
        push  AX            ;           NOT
        mov   AX,Data       ;
        mov   DS,AX         ;                    ALTER

;--Print STRINGY

Print_string_service:

        mov   ah,9          ;request PRINT STRING service
        mov   dx,offset STRINGY
        int   21h           ;invoke PRINT STRING service

;--Return to DOS

        ret

Dummy   EndP                ;END of Procedure
Code    EndS                ;END of code Segment
;--------------------------------------------------------

        END   Dummy         ;END of routine Dummy
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

STEP 4 (Use of ASM and LINK)
The only subtlety in the ASM/LINK process is that the "/m" option of the Linker must be used to produce a Map file for Periscope.

```
E>asm dummy;
The IBM Personal Computer Assembler
Version 1.00 (C)Copyright IBM Corp  1981

Warning Severe
Errors  Errors
0       0

E>link dummy

IBM Personal Computer Linker
Version 2.00 (C)Copyright IBM Corp 1981, 1982, 1983

Run File [DUMMY.EXE]:
List File [NUL.MAP]:  dummy.map/m
Libraries [.LIB]:

E>
```

The module Dummy.exe and the Map file Dummy.map have now been created.

STEP 5 (RUNning Periscope)

We are ready to use Periscope to debug our program Dummy.exe. We invoke the module run.com, supplied on the distribution disk, as follows:

```
E>run Dummy.exe
```

Periscope is now up and running. The remainder of this review describes its features, dispensing with any further "Steps."

A SAMPLE RUN

After typing "run Dummy.exe", we are greeted with:

```
AX=0000 BX=0000 CX=0180 DX=0000 SP=0100  BP=0000  SI=0000  DI=0000
DS=0C36 ES=0C36 SS=0C4A CS=0C46 IP=0000   NV UP DI PL ZR NA PE NC
                                WR SS:00FE = F346

                DUMMY:
0C46:0000 1E            PUSH    DS


Periscope's prompt is  ">" , and  typing "u" for
unassemble gives:

>u
                DUMMY:
0C46:0000 1E            PUSH    DS
0C46:0001 2BC0          SUB     AX,AX
0C46:0003 50            PUSH    AX
0C46:0004 B8480C        MOV     AX,0C48
0C46:0007 8ED8          MOV     DS,AX

                PRINT_STRING_SER:
0C46:0009 1E            MOV     AH,9
```

Notice the symbols DUMMY and PRINT_STRING_SER; the same symbols used in the source code. Well almost the same symbols. The label PRINT_STRING_SERVICE was truncated to PRINT_STRING_SER by Periscope, which retains only the first 16 characters of any Public name.

These labels are more than visual markers. One can "goto" them.The command G@PRINT_STRING_SER goes to the instruction corresponding to the label PRINT_STRING_SER:

```
0C46:0009 1E            MOV     AH,9
```

SETTING BREAKPOINTS

Periscope offers a rich set of primitives for setting breakpoints. Breakpoints may be set on code, memory, or registers.

To set a code breakpoint, use BC address , as in the following examples:

```
BC CS:100          or       BC @PRINT_STRING_SER
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

To set a memory breakpoint, use BM address address R and/or W. For example, to set a memory breakpoint from "start" to "finish," use:

```
BM @start @finish W
```

Any instructions that write (W) to this range of memory will cause a breakpoint to occur, before the instruction is executed.

To set a register breakpoint, use BR register test number . For example: BR CX EQ 0. This command causes a breakpoint when the value in CX is equal to 0. Register breakpoints will find valuable application in loop structures.

The BC, BR, and BM commands set "sticky" (present until erased) breakpoints.

To display any existing register breakpoints, use BR ?; to erase all register breakpoints, use BR *. A similar pattern holds for code and memory breakpoints.

Suppose that the following instruction, with address 0CDE:0, occurs in a loop:

```
MOV     AH,offset Printer_base[SI]
```

To set a sticky code breakpoint at this instruction, use BC 0CDE:0. The command G (Go) will stop at that address, and continue to do so for each iteration of the loop. The G command does not stop at register breakpoints, or other non-code breakpoints.

The GB (Go using Breakpoints) is similar to the G command, but it also stops at register breakpoints, memory breakpoints, and other non-code breakpoints.

DUMPING MEMORY

This is one of the most interesting aspects of Periscope. As an illustration of its power, the interrupt vectors located at 0000:-0000 are dumped using several display formats.

First, using the standard dump (D) command. The results are identical to those given by DEBUG.

```
>d 0:0
0000:0000  86 01 8C 0C 58 0B F0 BF-5D 0B F0 BF 62 0B F0 BF ...X.p?].p?b.p
```

Recall that the 8088 stores words in memory backwards, inspiring at least one writer to call them "backwords." So the first four bytes shown above indicate a segment:offset address of 0C8C:186 for interrupt 0.

Dumping the interrupt vectors using Periscope's double-word format gives:

```
>dd 0:0
0000:0000   0C8C:0186     BFF0:0B58     BFF0:0B5D     BFF0:0B62
```

Fantastic. Periscope unscrambled the "backwords" into "forewords" saving the programmer the need to mentally reverse the bytes.

Could things possibly get any better?

Yes!

The end result of using a template (also called a record definition) is shown below:

```
>dr 0:0 @int
Int_0:      0C86:0186
Int_1:      BFF0:0B58
Int_2:      BFF0:0B5D
Int_3:      BFF0:0B62
```

The addresses are not only neatly laid out, but are prefixed by an identifier, in this case the interrupt number, Int_x. How did I do it? Periscope allows programmers to make customized record definitions for the display of memory. The record definition, named INT, used in the creation of the above display was:

```
\INT              ; Interrupt Vectors
Int_0:,d,4        ; Interrupt 0 address
Int_1:,d,4        ; Interrupt 1 address
Int_2:,d,4        ; Interrupt 2 address
Int_3:,d,4        ; Interrupt 3 address
Int_4:,d,4        ; Interrupt 4 address
```

The command

```
>dr 0:0 @int
```

issued earlier means: display the area starting at 0:0, using the record definition whose name is INT. The "d" in the record definition indicates that the data is to be displayed in double-word format, and the "4" indicates that 4 bytes is required for the display of the data item. A semicolon indicates that the remainder of the line is a comment.

Periscope supplies record definitions for the following areas: PSP, FCB, and EXE header.

RESET BUTTON

A reset button is useful for breaking out of programs that have locked up the PC or its keyboard. The Submarine board contains a button that protrudes from the back of the PC. For

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

those who want greater convenience, a remote reset button is also provided in the Periscope package. But be forewarned — the button cannot always rescue the PC from the jaws of death. So keep the power on/off switch oiled!

HEXADECIMAL ARITHMETIC

Hexadecimal arithmetic and number-base conversions are supported.

```
To display a hex. number in decimal and binary, use the
"xh" command:

>xh fff0
FFF0h    65520d      1111 1111 1111 0000b

>xh -1
Error 10 - Invalid/missing number

To perform hex. arithmetic, use the "h" command:

>h 1 + d
000E

>h f*123
0000 110D


To translate a segment:offset address into a
physical address, use the "xa" command:

>xa a78:4
0A784H
```

SPLIT SCREEN

Suppose that a program X displays a panel on the screen. Using DEBUG, we can trace through program X, insuring that the top border was drawn correctly, then the bottom border, then the boundary, then the text in the panel. After DEBUG has displayed the top boundary, the next set of commands typically scrolls the top boundary display off the screen into never-never land. Periscope, on the otherhand, preserves this information on a "separate" or alternate screen. Pressing the F10 key toggles between the debugging screen and the alternate screen. The alternate screen shows the current stage of development of the panel.

The alternate screen approach, while useful, has at least two surprises.

Surprise 1. In the sample program Dummy, when I traced past INT 21h, I expected to see the string "I've got . . ." displayed on the screen. That is, when I travel from point A to point B, I expect to see the scenery along the way.

This string is displayed under DEBUG, but it is nowhere to be found on Periscope's screen. Actually, the string is not lost under Periscope, but appears on the alternate screen. While this is consistent with the function of the alternate screen, it is nonetheless surprising, as the anticipated string does not appear on the currently displayed screen.

Surprise 2. While tracing through an unassembly, Periscope quickly displays the alternate screen and then restores the debugging screen. This "flash" is somewhat hard on the eyes when you're furiously tracing through a program.

The above description applies to those having only one monitor. If your system has both a color and a monochrome monitor, you can debug on one monitor while simultaneously displaying the panel image on the other.

THE MANUAL

While explanatory, the tutorial will best be understood by those familiar with DEBUG's command language. For those not familiar with DEBUG, it will be a puzzling exercise in entering commands.

The manual may confuse first-time readers. For example, on page 4-2 of the tutorial section, reference is made to a "record

$35 DISK CACHE/PRINT BUFFER

Disk Cache

-Look ahead background disk reads
-Background disk writes
-Automatic detection of disk removal

Print Buffer

-Use inexpensive computer RAM to replace expensive hardware print buffer
-Smooths background printing

Plus

-User specified buffer size (10K - 600K)
-Dynamic partitioning of buffer to the disk cache and print buffer.

$25 DEBUG ENHANCER

-Set breakpoint at current instruction
-Use register names in command line
-Attach data line to register display
-One-key trace for int, call, rep & loop
-No machine hang when tracing

To order call (503) 883-3278
or write OREGON PROGRESSIVE SOFTWARE
1965 Manzanita Street
Klamath Falls, OR 97601

VISA or **MASTER CARD** Accepted. Mailed orders must indicate DOS version number (1.1, 2.0 or 2.1) and machine type. Extra charge for C.O.D. and orders outside the continental U.S. We pay domestic shipping and handling.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MEMO: C Programmers

QUIT WORKING SO HARD.

These people have quit working so hard: IBM, Honeywell, Control Data, GE, Lotus, Hospitals, Universities & Government Aerospace.

THE GREENLEAF FUNCTIONS™

THE library of C FUNCTIONS that probably has just what you need . . . **TODAY!**

. . . already has what you're working to re-invent
. . . already has over 200 functions for the IBM PC, XT, AT, and compatibles
. . . already complete . . . already tested . . . on the shelf
. . . already has demo programs and source code
. . . already compatible with all popular compilers
. . . already supports all memory models, DOS 1.1, 2.0, 2.1
. . . already optimized (parts in assembler) for speed and density
. . . already in use by thousands of customers worldwide
. . . already available from stock (your dealer probably has it)
. . . It's called the **GREENLEAF FUNCTIONS.**

Sorry you didn't know this sooner? Just order a copy and then take a break — we did the hard work. Already.

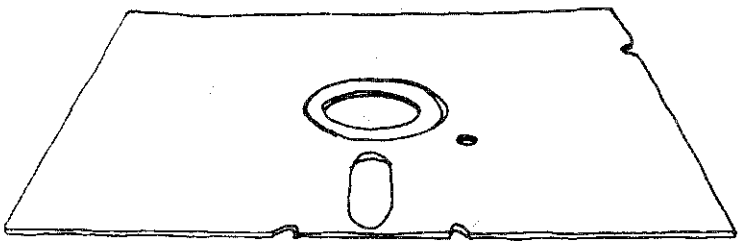
THE GREENLEAF FUNCTIONS GENERAL LIBRARY: Over 200 functions in C and assembler. Strength in DOS, video, string, printer, async, and system interface. All DOS 1 and 2 functions are in assembler for speed. All video capabilities of PC supported. All printer functions. 65 string functions. Extensive time and date. Directory searches. Polled mode async. (If you want interrupt driven, ask us about the **Greenleaf Comm Library.**) Function key support. Diagnostics. Rainbow Color Text series. Much, much more. **The Greenleaf Functions.** Simply the finest C library (and the most extensive). All ready for you. From Greenleaf Software.

. . . **Specify compiler** when ordering. Add $7.00 each for UPS second-day air. MasterCard, VISA, check, or P.O.

- **Compilers:**
  - CI C86 . . . . . $349
  - Lattice $395
  - Mark Williams . . . $475
- **General Libraries $175**
  (Lattice, Microsoft, Mark Williams, CI C86)
  (DeSmet C) $150
- **Comm Library $160**

GREENLEAF SOFTWARE

GREENLEAF SOFTWARE, INC.
2101 HICKORY DRIVE • CARROLLTON, TX 75006 • (214) 446-8641



definition," while the concept of record definitions is not explained until page 5-6.

A sample program is included in the Periscope package to illustrate the use of Periscope, but the information about Publics and the "/m" Linker option is somewhat scattered about the manual. An all-in-one-place, "start-to-finish" example would be useful.

Once you have mastered the basics of using Periscope, the value of the manual will increase.

GENERAL REMARKS

1. It is not possible to code breakpoints in ROM or unmodifiable code.

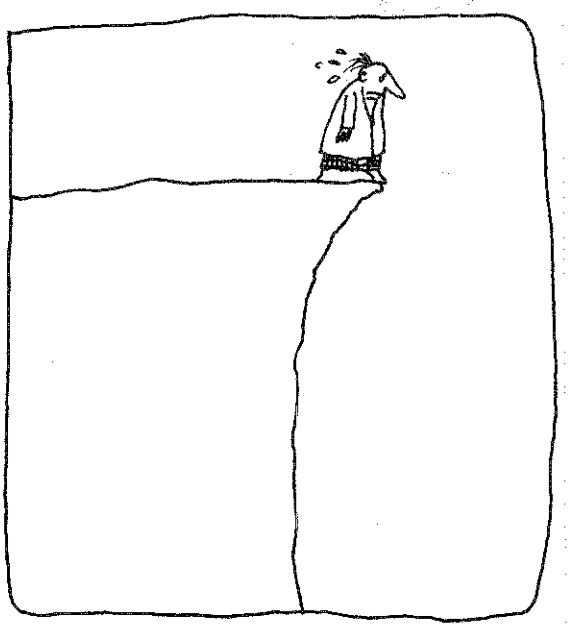
2. I could not redirect the output of Periscope to a file.

3. Periscope supports commands such as "D ES:DI". Contrast this with DEBUG, where one must enter the numeric value contained in the DI register.

4. Periscope does not support 8087 opcodes, whereas DEBUG does.

CONCLUSION

This product is a reliable and useful tool for any program mer's workshop. A number of innovative, helpful features are yours for the taking. □



Programmer at Breakpoint

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BOOK REVIEWS

by J. Edward Volkstorf, Jr.

Writing Efficient Programs by Jon Louis Bentley

One of the worst gloom and doom problems in programming is the program that solves the problem, works correctly, but runs like molasses being poured in winter. Or maybe worse, it eats up memory or disk space like there's no tomorrow. In any event, most professional programmers usually have what I call a "bag of tricks" that they use in optimizing their programs to get them working at some acceptable level of performance.

Occasionally you will find magazine articles or proceedings at conferences where someone's horror story and their solution are recounted. Some of these are of the "How to Squeeze 100 Nanoseconds out of the MOV instruction" variety: interesting reading, but very narrow in scope.

You might think a collection of these techniques would make for a good book. Well think no more, because Mr. Bentley has done the job with his book "Writing Efficient Programs" published in 1982 by Prentice-Hall. It is a collection of 27 tried and true techniques that have wide applicability in making programs more efficient.

To use Mr. Bentley's words,

The book is not intended as a scholarly treatise on esoteric aspects of computer science. Rather, its purpose is to give practicing programmers both a mind-set and some practical tools to help them make their programs more efficient.

All of the tools and techniques are very general in nature and not specific to a particular programming language or computer. Furthermore, they are illustrated in numerous forms, with many having some comparative numerical measures of their effect. For the most part, a dialect of Pascal is used to illustrate, but there are some C coding examples and one assembly language example.

Each chapter concludes with a set of suggested exercises. Although there are no solutions given, the exercises give the reader ample food for thought in applying the techniques. Of course most readers will no doubt have some programs of their own for putting the techniques to use.

Seven pages list references to other books and articles on the subject of efficient programming. Many of the references are follow-ups to the techniques illustrated in in the book.

The techniques are organized in three categories:

1. Modifying data structures — space for time and time for space trade-off issues are explored.

2. Modifying code — rules for loops, logic, procedures, and expressions are discussed.

3. Fundamentals — techniques that can be applied to systems as a whole (versus the previous groups which are for coding segments) are considered.

The book contains a discussion of why to optimize, when and when not to do so. A chapter is included that applies the techniques to the Quicksort algorithm and the binary search. This is an excellent case study of the situation where, once you are given an algorithm, here's how you might go about applying some of the efficiency techniques.

For example, after the binary search is initially written, five fragments from it are examined in detail. This section then concludes with an elegant binary search program by Knuth that has been optimized for 1000 items.

The book is best read in its entirety at a moderately rapid pace for a quick overview. The sections and techniques you find the most interesting can then be reread and studied more closely. Although rather short in size at 170 pages, it is quite meaty in content. It also includes four appendices that contain material on optimizing an assembly language problem, a review of the Pascal dialect used, and a summary of the rules developed in the book.

Most veterans in this field know that there is more to the art of 0programming than just coding programs using some language. Knowing how to code well is truly an art form which you learn primarily by doing and re-doing until your skills are honed to a fine level. There is also the process of learning from other masters of the craft. This book is a sharing of the skills learned by these masters.

All in all, I wholeheartedly recommend the book to the serious hobbyist, and the professional programmer. More books of this type are needed.

Writing Efficient Programs
by Jon Louis Bentley
Copyright 1984
Published by Prentice-Hall, Englewood Cliffs, New Jersey
Price: $24.95

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Bluebook of Assembly Routines for the IBM PC & XT
by Christopher L. Morgan

Part of learning a new programming language is reviewing how seasoned veterans use the language. Every language has an inherent outlook and underlying style that sometimes take a while to discern when you initially use the language. There's also the need for subroutines and functions that get you up to speed faster than if you were to develop them yourself. It was with these two thoughts in mind that I purchased the books "Bluebook of Assembly Routines for the IBM PC and XT" by Christopher L. Morgan and "C Programmer's Library" by Jack J. Purdum, et al.

I had been studying both languages and was beyond the "learn the syntax" stage. So it was time to get down to writing some code and seeing how the pro's do it. A good source for this is of course magazines such as "Programmer's Journal" and others; but I wanted many big chunks to work with, hence these two books jumped out at me at the bookstore.

They both have been good investments, but each could offer more and, in some ways, in a better format. First the "Bluebook." This volume is published by The Waite Group, headed by Christopher Waite, which has brought you many notable books on the PC, graphics, DOS, CP/M and UNIX (among other subjects). Their books are characterized by well-written text, nice crisp, clean page layouts, and ample 3D-oriented illustrations.

The "Bluebook" is no exception. It consists of 244 pages of IBM PC, 8088/8086, assembly language organized in nine chapters with a total of 103 subroutines. They consist of I/O routines, Binary, BCD and Floating Point Conversions, Multi-digit Arithmetic, Graphics, Sound, Strings, and File Manipulation. Many routines are fairly short and simply illustrate how to make DOS and BIOS calls. Others are two and three pages long and provide some useful tools, such as counting words in a file, converting a file to WordStar format, searching a list of strings, and displaying graphics mode "stroke" characters.

Many (if not most) of the routines work together, building upon each other. For example, a single-character standard output routine is provided, and later a standard message output (multi-character) routine is described. The latter uses the single-character I/O routine. Thus what you are getting is really a illustrated library of useful routines with techniques for segmenting and organizing your code.

Each routine begins with a standard set of headings that consist of the routine's name, function, what it expects upon entry (or input), what it outputs, the registers used and modified, which segments are referenced, any other routines called, and a closing special notes section. As might be expected, each routine is also fully annotated, with each line of code containing a trailing comment.

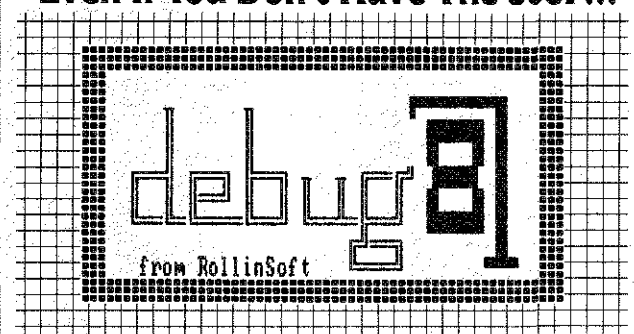
The opening chapter is an introduction to how the routines can be used. It also provides an overview of the 8088's registers and how they are used with the four segment registers. Another section describes integrating the routines with other assembly language programs as well as calling them from BASIC.

I feel this is an excellent book to help you learn the PC's assembly language and also as the source and start of your own assembly language library. Pair this book with an introductory text and you will be well on your way to learning the ins and outs of assembly language programming.

My only disappointment was that some of the routines were a little skimpy on substance. For example, the standard output of a space and the standard output of a CRLF were used in some other routines, so they were provided as separate routines in their own right. There were no printer I/O routines. This seems somewhat of an oversight since the display, disk and comm line I/O routines were provided. Why not make a complete set?

Another more glaring fault was the lack of any error checking in most routines. This is mentioned though, as in the case of

Even If You Don't Have The 8087...



➪Extends & enhances the power of DEBUG by adding complete support for the 8087 numeric coprocessor.
➪Register values are displayed with 18-digit floating-point accuracy.
➪Status, Tag, and Control words are identified.
PLUS!
➪Enter decimal values directly into any 8087 register. Initialize the 8087 with one command.
➪Requires DOS 2.x DEBUG.

New Commands

K clear screen, set screen mode, attribute, and border
B binary and decimal conversions
H Hexarithmetic with 6 operations
Y one-step trace of CALL, INT, LOOP and REP. One-line trace display.
X 8087 npX display and modify
* timer-interrupt, FWAIT bugs fixed

Rush $49.95 to:
RollinSoft PO Box 12147, La Crescenta CA 91214
Created by Dan Rollins

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the graphics routines that do not check the screen coordinates passed to them. (By the way, all the graphics routines were in the four-color, medium-resolution mode — no high-level resolution graphics.) Some error processing was illustrated for opening disk files, but nothing on the checking of bit flags in status words.

A good place to provide some error processing would have been in the comm I/O or the missing printer I/O routines. Possibly the difficult subject of error processing could have been illustrated with the addition of an error routine that could be called for a screen display of the error reason.

Also, no floating point arithmetic operations are provided. Conversions to and from the standard floating point format are given and ASCII string integer operations are available, but not for floating point.

Obviously you could go on and on about what else would be useful to have. The net balance is that the book is a good one, and anyone just beginning to learn to use assembly language is well advised to get it.

Bluebook of Assembly Routines for the IBM PC & XT
by Christopher L. Morgan
Copyright 1984 by the Waite Group
Published by New American Library, New York
Price: $19.95 (No book/disk software combination was listed)

Computer Game-Playing, Theory and Practice
edited by M. A. Bramer

This 300+ page book is a collection of 21 articles that for the most part would be of interest primarily to the advanced computer scientist or artificial intelligence researcher. The book is one of a number of texts published as part of the Ellis Horwood Series in Artificial Intelligence. The "theory" part of this book dominates, so those not versed in the rigors of AI may find the going pretty difficult. Nevertheless, a wide variety of concepts is covered, with some interesting games.

Half of the book consists of articles on the games of chess and GO. For chess the articles consider various techniques for move searches and selection, machine learning, and in particular the problems in handling the end game. Only one article examines the general problem of writing a program to play the game. Its algorithms are expressed in the LISP language.

The game of GO, although simpler in concept, far exceeds chess in the number of moves possible. A player has from 20 to 30 potential moves to select from at each stage of chess, whereas a GO player has from 200 to 300 possible moves. Brute force search techniques are of very limited use in computerizing the game. Hence research in GO is more in the area of representing the knowledge needed to play the game and pruning the possible moves to a manageable size.

Lumberjack and Logged Disk Drive

Now With Flipscreen

SYMD
SYMBOLIC•DEBUGGER

Faster, easier more productive

SYMD is a unique new programming aid that dramatically reduces the time and effort needed to identify and correct programming errors. SYMD simplifies debugging by utilizing source code symbols and line numbers wherever an address or offset is required. Flipscreen feature allows separate display, including graphics, for SYMD and program under test. SYMD also lets you: Examine and/or change real number data; assign permanent or temporary breakpoints, including pass counts to control looping; check instruction execution sequences with backtrace commands; use profiling commands to identify the most active parts of a program. And much more. SYMD can be used with compiler or assembly language programs. System requirements: PC-DOS™ or MS-DOS™ 1.1 or 2.0; 192K recommended; 80-column display.

SYMD is priced at only $125. Free literature on request. VISA or MasterCard accepted.

™MS-DOS is a trademark of Microsoft Corporation. ™PC-DOS is a trademark of the IBM Corporation.

D+V Systems
400 Amherst Street Nashua, NH 03063 (603) 881-7140

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Four articles look at the game from various perspectives. Programs that play GO lag far behind the ability of those that play chess.

A third section of articles looks at other games, with two articles on othello, two on scrabble, two card games, and one on 3 cushion billiards. The latter seems an unlikely candidate for computer game playing. The article describes a model of the game of 3 of 3 cushion billiards (not to be confused with the more popular pocket billiards game played in the USA). The shots selected by the model for a particular arrangement of balls are compared to those an experienced player might select. The comparison supposedly provides some insight into the nature of human versus machine intelligence.

The Scrabble game articles are interesting. The first article considers the problem of developing a word-value equation that assigns some value to each potential word that can be played, based on the current set of letters available to the computer. From this a playing strategy evolves. For example, whether or not to hold the letter U for later use in a word that contains the letter Q needs to be considered. Designing an algorithm that allows selection and searching of potential words with the highest point value is discussed in the second article. The article describes a lexicon tree of potential word choices and how the tree is searched.

The book closes with three general articles on game tree searching. The concept of alpha/beta pruning of trees is illustrated and contrasted with brute force techniques. The game of checkers is also examined in this section from the standpoint of the potential for parallel processors using the alpha/beta technique out-performing a single processor.

Even with the complexity of the material, most of the articles are quite interesting. Admittedly, much of the subject matter was over my head, but I was able to get a better idea of some mainline AI concepts from the book. My major complaint is that the learning experience it offers is not in line with the book's cover price of over $64.

Any reader looking for an introduction to computer game playing would be better off looking elsewhere. For the advanced reader there should be enough material here to make the book a worthy investment.

Computer Game-Playing, Theory and Practice
edited by M. A. Bramer
Copyright 1983
Published by Ellis Horwood Limited
Chichester, West Sussex, England
(Distributed in USA by John Wiley & Sons, New York)
Price: $64.95

C Programmer's Library
by Jack J. Purdum, Timothy C. Leslie and Alan L. Stegemoller

The "C Programmer's Library" is based on generic C and is not slanted towards the IBM PC, or any other micro for that matter. Of course this is one of the goals of C: machine independence. Nevertheless, the very last appendix in the book notes that routines were tested on eight different compilers under CP/M, MS-DOS, and APPLE DOS. It was simply noted that "test results varied widely" (!). Whatever that's supposed to mean is never described.

In any event, the book consists of 366 pages of C code that could have been better organized. Two opening chapters (numbered 0 and 1 — a chapter 0 appears to be the vogue thing nowadays in computer books) introduces some general thoughts on programming in C, developing a function library, and describing C data types. This is followed by four chapters of C code and assorted functions, ending with a chapter that is a fairly large C program for cataloging books.

The four chapters of C code consist of one on sorting, another on creating a general terminal library (for I/O to ASCII teletype-like terminals), a handful of "code fragments" — various list structures such as linked list and binary trees — and lastly a 100-page ISAM routines chapter.

Part of my problem, I suppose, was my personal expectations. I was thinking that here was printed copy of the source for (pick one) the Blaise Computing, Green Leaf, Lattice, etc. functions. Well, it's not even a subset.

Another disappointment was the lack of organization for each routine. By comparison, the assembly language "Bluebook" had a standard introduction to each routine. Not so in the C book. Each function or fragment just followed the other as in any prose or book. Headings with variables in, variables out, etc., would have been very useful.

The terminal I/O library seems only moderately useful to me for IBM PC programmers. This is especially the case with the standard libraries that most C languages come with and the ANSI.SYS option in DOS 2.0. The ISAM routine is overwhelming — and worst of all is admittedly incomplete. The delete routine is missing but can be bought (along with all the source code) on a diskette for about $125. I have a feeling that one of the B-tree routines such as PHACT or Btrieve might be better choices.

So what does this leave? A chapter on sorting and one on fragments. There is also a five-page supplement of errata (in the March 1984 first printing), all of which is for the C source code. Sorry, Charlie, the "C Programmer's Library" is a disappointment.

C Programmmer's Library
by Jack J. Purdum, Timothy C. Leslie, and Alan L. Stegemoller
Copyright 1984
Published by Que Corporation, Indianapolis
Price: $19.95
(Book/disk software combination $124.95) □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NEW C JOURNAL

The noted writer, consultant and C Standards committee member, Rex Jaeschke, is heading up a new journal for the C language. The first issue will come out in April and it will cover a multitude of C topics, certainly one being the upcoming standardization of the C language. For more information call the publishers and ask about their new C Journal.

The C Journal
InfoPro Systems
ph. (201)989-0570

MICRO/CICS

If we all had our druthers,we, of course, would like to spend the rest of our lives in CICS and IMS environments. Now is your chance to do just that on an IBM PC XT/370. Unicorn Systems Company has announced MicroCICS, a menu-driven development tool which supports IBM's CICS application development on a desktop IBM PC XT/370. Programmers can enter, compile, test and debug CICS programs without the necessity of accessing the mainframe until their complete code is ready to upload. MicroCICS allows entry, editing and compilation of command-level CICS source programs, online screen generation and BMS maps, creation of test files, program execution and testing, and debugging tools and host communications. The first copy of MicroCICS is $4,495.

Unicorn Systems Company
3807 Wilshire Blvd.
Los Angeles, CA 90010
(213) 380-6974

BORLAND MEETS COMPUSERVE

Borland's president spent so many hours on CompuServe answering questions about Turbo Pascal and Sidekick that they have decided to start a BOR SIG. Users can download sample programs and send messages to other Borland programmers. By the way, over 125,000 copies of Turbo Pascal have been sold.

SCIENTIFIC CALCULATOR & ASYCH MANAGER

Blaise Computing is marketing a memory-resident scientific programmable calculator (SPARKY). It captures the functionality of the best handheld desktop calculators and gives them to you instantaneously. It can read and write files of programs or data. A Hewlett-Packard calculator model is used for displaying the stack and user-defined memory variables. SPARKY accepts input numbers in any base from 2 to 35 and generates output in standard real, scientific real, hex, octal or binary format. The 8087 can be accessed if you have one installed.

A second product for system developers is ASYCH MANAGER. It gives you interrupt-driven and polling control over communications ports from within your high-level program. Use ASYCH MANAGER as the major tool for file transfer modules, realtime data acquisition, and printer/plotter control. The package includes a manual and is available in C and Pascal versions.

AYSCH MANAGER, $175
SPARKY, $75
Blaise Computing
2034 Blake St.
Berkeley, CA 94704
(415) 540-5441

THE MODULA-2/PASCAL BOOK

You have seen the C for BASIC programmers book, and Pascal for BASIC programmers book. Now there is a Modula-2 for Pascal programmers book. The text explains concepts that are not found in Pascal modules, separately compiled modules, programs and subprogram modules, utility modules, module library, low-level programming, coroutines, interrupts, and procedure variables. The author, Richard Gleaves, has written a number of highly recommended tutorial manuals. The book is priced at $15.95.

Modula-2 for Pascal Programmers
Springer-Verlag New York Inc.
175 Fifth Ave.
New York, NY 10010
Attn: Louis Bavaro
(212) 460-1601 □

ADVERTISERS

- ABComputing 11
- Blaise Computing 33
- Carousel Microtools Inc. 31
- Cybernetic Micro Systems 18
- D&V Systems 41
- Data Base Decisions 9
- DWB Associates 13
- Essential Software 27
- Greenleaf Software 38
- Morgan Computing 35
- O.E.S. Systems 8
- Opt-Tech 25,30
- Phoenix Computer Products . . . . . Cover/Front
- Programmers Shop 5
- Qualitas 29
- Rational Systems Inc. 6
- RollinSoft 40
- Smartware 37

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ARE YOU AN ISSUE SHY OF A FULL SET?

Complete your library! Order any back issues you may have missed — or become a new subscriber. Use the form below or the handy subscription envelope provided. You'll be glad you did.

- Vol. 1, No. 1: Norton Pascal I/O, Program Style Manual, COBOL, System Analysis, BASIC modules, Rounding error.
- Vol. 1, No. 2: Defining character sets, C86 BIOS calls, Copy protection, Pascal code, Input routines, Writing an Assembler.
- Vol. 1, No. 3: ASM dumb terminal, C source code, ASM graphics line drawing, Text editors reviewed, Utilities.
- Vol. 1, No. 4: Diskette directory via Pascal, MASM WHILE-WEND, C/ASM interface, ASM stay resident routine, Structured programming.
- Vol. 1, No. 5: Multi-tasking software review, Transporting C Programs, Programmers calculator, dBase bargraphs, Utilities.
- Vol. 1, No. 6: ASM printer configuration, DOS patch utility, Hi-speed graphics, Software ownership, dWINDOW review.
- Vol. 2, No. 1: Freelance consulting, Sorts in C, Pascal text editor source code, Designing national database, Ultra utilities.
- Vol. 2, No. 2: Understanding C through Assembly language, Utility routines in C and Assembly, Music in C (using a C music program), Cornucopia of software development tools.
- Vol. 2, No. 3: Software development environments, Creating a productive screen handling tool, Generating graphics within windows, Create, edit and play your musical ideas, Linking Assembler to Pascal and Basic.
- Vol. 2, No. 4: Writing a custom ROM BIOS for PC-DOS, The future of disk protection, Time-saving MASM Macros, C routines you can purchase.
- Vol. 2, No. 5: Scroll control using BASIC and Assembler, RTool, a PC-resident programmer's toolkit, Calling Assembly routines from dBase II, Pascal's pointer power.
- Vol. 2, No. 6: Periscope-Building a hardware board for the IBM PC, AT&T surprises all, dBase II to dBase III?, Intel 80286 bugs by FNE, calendar functions for BASIC.

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Telephone _____

Send me the following back issues:
$5.50 each U.S., ($6.50 Canada, $8.50 Foreign, U.S. funds) pre-paid only

☐ Vol. 1, No. 1	☐ Vol. 1, No. 5	☐ Vol. 2, No. 3
☐ Vol. 1, No. 2	☐ Vol. 1, No. 6	☐ Vol. 2, No. 4
☐ Vol. 1, No. 3	☐ Vol. 2, No. 1	☐ Vol. 2, No. 5
☐ Vol. 1, No. 4	☐ Vol. 2, No. 2	☐ Vol. 2, No. 6

☐ 1 year bi-monthly subscription:
$24.00 ($30 Canada, $40 Foreign, U.S. funds only)

☐ Please renew my subscription

☐ Check enclosed ☐ Bill me (subscriptions only)

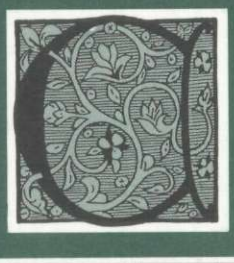
☐ Charge my Bankcard: ☐ VISA ☐ MC ☐ Am Ex

_____ Expir. Date _____

Signature _____

Send to:
PROGRAMMER'S JOURNAL
P.O. Box 30160
Eugene, OR 97403
(503) 345-3043

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



November's Comdex

C—Its Uses and Abuses

Using the C Preprocessor

Memory Segmentation Examples

Index Files (Part I)

Structured Programming Language

Periscope, a Symbolic Debugger

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED